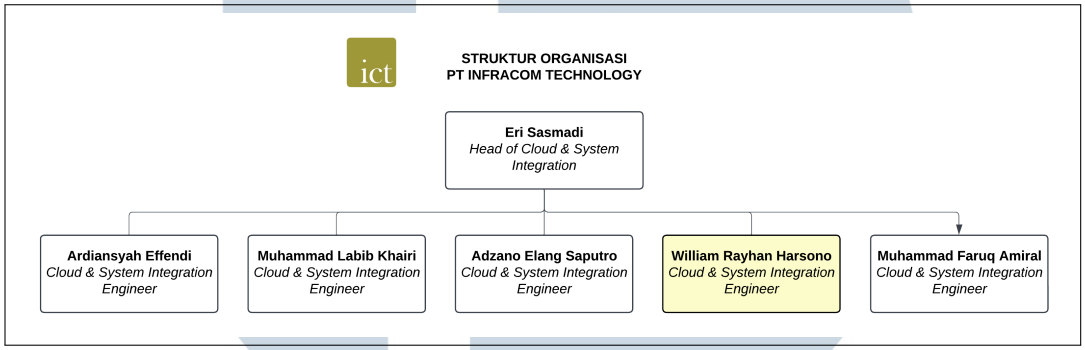
BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Organisasi

Selama program kerja magang di PT InfraCom Technology, posisi yang diberikan adalah sebagai anggota tim *Cloud & System Integration Engineer*.



Gambar 3.1. Struktur organisasi perusahaan PT Infracom Technology

Dalam gambar 3.1 ini, tanggung jawab mencakup dukungan terhadap integrasi infrastruktur *cloud* perusahaan, peningkatan efisiensi sistem, serta memastikan kelancaran layanan teknologi informasi yang mendukung berbagai aktivitas bisnis perusahaan.

3.1.1 Media yang Digunakan dalam Proses Magang

Dalam proses magang, berbagai media digunakan untuk mendukung kegiatan dan mempermudah komunikasi, yaitu:

1. Google Meet → digunakan sebagai sarana untuk *share screen* saat pertemuan mingguan yang dilakukan secara *offline*, memudahkan penyampaian materi dan kolaborasi visual dengan tim.
2. WhatsApp → digunakan sebagai *platform* komunikasi utama untuk koordinasi harian dan pertukaran informasi antar anggota tim, memastikan respons yang cepat dan efisien terhadap perubahan situasi.

3. Google Spreadsheet → digunakan untuk pencatatan dan pelaporan *daily task* yang dikerjakan selama magang, memungkinkan pengawasan progres secara terstruktur dan transparan oleh *supervisor*. Selain itu, digunakan untuk menampilkan daftar tugas yang perlu dikerjakan serta status tugas yang sudah selesai, sehingga memudahkan manajemen waktu dan prioritas kerja.

Selain media komunikasi tersebut, berbagai alat lain seperti *Terraform* dan *Ansible* juga digunakan untuk mendukung pengelolaan infrastruktur cloud dalam lingkungan *Hybrid Multi Cloud*. Penggunaan alat ini memungkinkan otomasi dan manajemen infrastruktur yang lebih efektif, serta meningkatkan kolaborasi dalam tim saat menghadapi tantangan teknis.

3.2 Tugas yang Dilakukan

Tugas utama mencakup implementasi solusi *best practices* dalam pengelolaan infrastruktur *cloud* menggunakan *tools Infrastructure as Code* (IaC) seperti *Terraform* dan *Ansible*, termasuk penerapan melalui *Ansible Tower* untuk memastikan *provisioning* berjalan efisien dan sesuai standar. Dalam lingkup *Oracle Cloud Infrastructure* (OCI), kegiatan meliputi *provisioning* dan pengelolaan layanan seperti *Autonomous Database* (ADB) dan *Autonomous Container Database* (ACD), serta mendukung operasional *Exadata Cloud@Customer* untuk memenuhi kebutuhan klien. Penerapan *Landing Zone* dilakukan dengan mengelola tiga *environment* utama, yaitu *Development*, *Test*, dan *Production*, menggunakan pendekatan holistik yang mencakup tahapan *assessment*, analisis, hingga strategi migrasi. Selain itu, pengelolaan *AWS Landing Zone*, *AWS Organizations*, serta *Account Management* difokuskan pada implementasi *pre-configuration* dan *guardrails* melalui *AWS Control Tower* sesuai dengan standar yang berlaku. Untuk mendukung aktivitas tersebut, diagram infrastruktur dan presentasi disusun guna menjelaskan konsep seperti *Organizational Unit* (OU), *Service Control Policies* (SCP), *billing*, serta *monitoring* dalam berbagai proyek yang dikelola.

Alasan utama di balik penerapan *Infrastructure as Code* (IaC) menggunakan *Terraform* adalah untuk meningkatkan efisiensi operasional dan standarisasi dalam pengelolaan infrastruktur *cloud* di PT Infracom Technology. Proses *provisioning* dan konfigurasi sumber daya *cloud* sering dilakukan secara manual melalui antarmuka pengguna grafis, yang tidak hanya memakan waktu tetapi juga rentan terhadap kesalahan manusia, seperti inkonsistensi konfigurasi dan kelalaian dalam pengaturan keamanan. Dengan mengadopsi *Terraform*,

perusahaan bertujuan untuk mengotomatisasi proses *provisioning*, mengurangi potensi kesalahan, dan memastikan konsistensi *deployment* di berbagai *environment* seperti *Development*, *Test*, dan *Production*. Selain itu, penggunaan *Infrastructure as Code* memungkinkan dokumentasi yang lebih baik melalui kode yang dapat dibaca dan diulang, sehingga memfasilitasi kolaborasi antar tim dan mempermudah proses *troubleshooting*. Pendekatan ini juga mendukung skalabilitas yang lebih baik, karena infrastruktur dapat ditingkatkan atau dikurangi sesuai kebutuhan dengan cepat dan efisien. Dengan demikian, penerapan *Terraform* sebagai bagian dari strategi menjadi langkah strategis dalam mengoptimalkan pengelolaan *cloud*, meningkatkan keandalan serta keamanan layanan di perusahaan, dan memastikan bahwa infrastruktur siap mendukung pertumbuhan bisnis.

Tabel 3.1. Ringkasan Tugas Mingguan (Juli - November 2024)

Minggu	**Ringkasan Tugas**
Minggu 1	Mempelajari layanan dasar *AWS* seperti *EC2, EBS, IAM* serta mengeksplorasi fitur keamanan dan manajemen database untuk memperdalam pemahaman infrastruktur cloud.
Minggu 2	Mengembangkan *assessment* untuk migrasi *data center* ke *cloud*, termasuk membuat layout arsitektur dan mendiskusikan strategi migrasi bersama tim.
Minggu 3	Melakukan finalisasi *use case* dan strategi migrasi yang diberikan oleh *supervisor*, mempelajari *compliance* dan regulasi *OCI*, mengikuti pelatihan *ransomware protection* di *AWS*, serta mulai mendalami *Terraform*.
Minggu 4	Menyusun artikel teknis terkait layanan *AWS* terbaru, menghadiri *DevOpsDays Jakarta*, dan berpartisipasi dalam *meeting* untuk instalasi *Oracle DB* di *Docker*.
Minggu 5	Menyelesaikan dokumen *ICT Foundational Technical Review for SPP Self-Assessment* untuk evaluasi internal dan mengikuti diskusi proyek bersama tim terkait proyek klien.
Minggu 6	Mengikuti Meeting terkait Merancang *Proof of Concept (PoC) ExaCC* untuk klien, mempersiapkan ujian *AWS Solutions Architect*, dan mengikuti pertemuan untuk desain infrastruktur dengan tim proyek.
	Lanjut ke halaman berikutnya

Minggu	Ringkasan Tugas
Minggu 7	Mengikuti pelatihan dan ujian untuk sertifikasi *Oracle Generative AI Foundational Associate* dan memulai eksplorasi konsep *Infrastructure as Code (IaC)* dengan *Terraform*.
Minggu 8	Melanjutkan pembelajaran *Terraform*, menciptakan skenario otomatisasi di *AWS* dengan *Terraform*, dan mencoba provisioning *EC2* untuk meningkatkan pemahaman praktis.
Minggu 9	Melakukan persiapan dan presentasi *sharing session* tentang *Terraform*, mencakup *best practice* penerapan *IaC* untuk *AWS*, serta melakukan tinjauan ujian *AWS SAA*.
Minggu 10	Menyusun dan menyelesaikan dokumentasi *Terraform* yang komprehensif dan ikut serta dalam rapat terkait provisioning *AWS* untuk mendukung proyek perusahaan.
Minggu 11	Menyelesaikan dokumentasi *Terraform*, melaksanakan *onsite provisioning* untuk klien, serta melanjutkan persiapan intensif untuk ujian *AWS Solutions Architect* melalui *Udemy*.
Minggu 12	Melakukan persiapan ujian *AWS Solutions Architect Associate*, berdiskusi tentang solusi *ExaCC* dengan klien, dan mengerjakan soal ujian sebagai bagian dari persiapan sertifikasi.
Minggu 13	Mengeksplorasi *Oracle Autonomous Database* dan konsep *multitenancy*, serta mempelajari *Ansible* untuk mendukung manajemen *hybrid multi-cloud*.
Minggu 14	Mempelajari dasar-dasar *Ansible* melalui dokumentasi dan menyusun dokumentasi *Ansible* yang akan digunakan oleh tim internal.
Minggu 15	Mempelajari lebih dalam tentang *Ansible Automation Controller* dan memberikan presentasi singkat kepada tim untuk meningkatkan pemahaman bersama.
Minggu 16	Mengikuti *Proof of Concept (PoC) ExaCC* untuk klien, menyelesaikan dokumentasi *Ansible*, dan membuat proyek demo automasi untuk menunjukkan aplikasi praktis *Ansible*.
Minggu 17	Mengeksplorasi terkait *job isolation* di *Linux* menggunakan *bubblewrap*, serta mengonfigurasi ulang *Ansible Playbook* agar mendukung *OCI Landing Zone*.

Lanjut ke halaman berikutnya

Minggu	Ringkasan Tugas
Minggu 18	Mengirimkan hasil provisioning *OCI Landing Zone* melalui *Ansible Tower* dan memverifikasi ulang proses *provisioning* di tiga lingkungan (*Development, Test, Production*).
Minggu 19	Melakukan presentasi hasil *provisioning OCI Landing Zone* kepada tim internal, mengeksplorasi konsep *compartment* dan *tenancy*, serta memulai *new task* untuk eksplorasi mendalam tentang *Landing Zone*.
Minggu 20	Mempraktikkan pembuatan *AWS Organization* termasuk struktur *Organizational Unit (OU)* dan manajemen akun, serta mempersiapkan presentasi akhir terkait *Account Management* dan *Pricing Relationship* di *AWS*.

3.3 Uraian Pelaksanaan Magang

Pada bagian ini, dijelaskan mengenai aktivitas yang dilakukan selama pelaksanaan magang di PT InfraCom Technology. Tugas utama yang dijalankan berfokus pada bidang *Cloud & System Integration Engineering*, di mana tugas tersebut meliputi proses desain, implementasi, serta pengelolaan infrastruktur berbasis *cloud computing*.

Uraian ini mencakup perangkat penunjang yang digunakan, penguasaan layanan dasar cloud computing melalui *platform* Amazon Web Services (AWS), serta pencapaian sertifikasi yang mendukung peran dan tanggung jawab yang diberikan. Dengan pendekatan sistematis dan terstruktur, seluruh proses pelaksanaan magang bertujuan untuk meningkatkan pemahaman teknis, kemampuan *problem-solving*, serta profesionalisme dalam bekerja di lingkungan teknologi informasi yang dinamis.

3.3.1 Perangkat Penunjang

Dalam menjalankan tugas sebagai *Cloud & System Integration Engineer*, terdapat beberapa perangkat keras dan lunak untuk mendukung kegiatan sehari-hari. Berikut rincian perangkat yang digunakan:

- **Perangkat Keras**
  - Macbook Pro M1 Pro

  - Chipset M1 Pro dengan 10-core CPU dan 16-core GPU
  - RAM 16GB
  - SSD 512GB

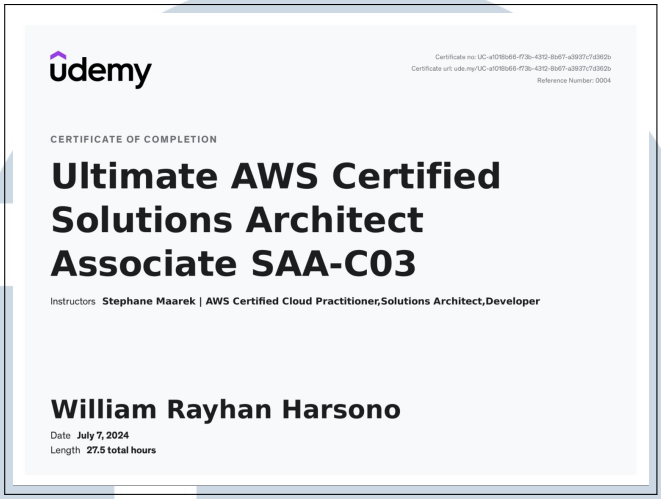
- **Perangkat Lunak**
  - MacOS sebagai sistem operasi utama
  - Terraform versi 1.5 - 1.9 sebagai *Infrastructure as Code* tool
  - Ansible versi 2.14 untuk otomatisasi konfigurasi
  - AWS CLI versi 2.12 untuk interaksi dengan layanan AWS
  - Visual Studio Code sebagai *text editor* utama

3.3.2 Penguasaan Layanan Dasar Cloud Computing

Pada tahap awal magang, fokus utama adalah memperkuat pemahaman mengenai layanan dasar *cloud computing*, khususnya pada *platform* Amazon Web Services (AWS). Untuk mencapai tujuan tersebut, pembelajaran dilakukan melalui serangkaian kursus daring di *platform* Udemy. Materi pembelajaran mencakup *AWS Certified Solutions Architect - Associate (SAA-C02) Complete Course*, *AWS Essentials for Beginners*, serta *Introduction to Cloud Computing with AWS*. Berbagai konsep dan layanan dasar AWS dipelajari melalui kursus ini, antara lain *Amazon EC2 (Elastic Compute Cloud)* sebagai layanan komputasi yang menyediakan server virtual di *cloud*, *Amazon S3 (Simple Storage Service)* yang menawarkan penyimpanan objek skalabel dan aman, *Amazon RDS (Relational Database Service)* untuk pengelolaan basis data relasional secara terkelola, *AWS IAM (Identity and Access Management)* yang memungkinkan pengaturan akses dan izin pengguna, serta *Amazon VPC (Virtual Private Cloud)* yang menyediakan jaringan virtual terisolasi.

Pembelajaran melalui kursus-kursus tersebut memungkinkan pemahaman prinsip-prinsip dasar *cloud computing* secara mandiri dan fleksibel. Materi yang disajikan bersifat komprehensif, dilengkapi contoh praktis, sehingga memudahkan penerapan langsung atas apa yang telah dipelajari. Gambar 3.2 menampilkan sertifikat dari Udemy, sebagai bukti penyelesaian materi yang sekaligus memperlihatkan kompetensi dalam memahami layanan dasar AWS. Dengan pengetahuan ini, penguasaan dalam implementasi dan pengelolaan infrastruktur

cloud selama magang menjadi lebih optimal, serta mampu berkontribusi secara efektif dalam tim *Cloud & System Integration Engineer*.



Gambar 3.2. Sertifikat digital dari Udemy

3.3.3 Pencapaian Sertifikasi AWS Solutions Architect - Associate (SAA-C03)

Topik ujian yang diujikan dalam sertifikasi ini mencakup berbagai aspek krusial bagi kesuksesan implementasi *cloud computing* secara menyeluruh. Pertama, konsep *resilient architecture* menjadi pusat perhatian, di mana kandidat diharapkan mampu merancang infrastruktur yang elastis, tangguh, dan tahan terhadap kegagalan sistem. Pendekatan ini memastikan layanan dapat tetap beroperasi secara optimal meski menghadapi fluktuasi beban kerja maupun gangguan teknis.

Selanjutnya, fokus pada solusi berperforma tinggi menuntut kemampuan untuk menentukan jenis layanan, konfigurasi, serta pola penerapan yang paling efektif agar aplikasi dan sistem dapat memenuhi kebutuhan bisnis secara konsisten. Kinerja infrastruktur yang baik akan mempengaruhi pengalaman pengguna, sekaligus mendukung pencapaian target operasional perusahaan.

Di sisi lain, desain yang aman dan terawat (*secure and operationally excellent design*) menekankan penerapan prinsip keamanan, pengelolaan identitas, serta pengawasan berkelanjutan dalam lingkungan AWS. Aspek ini mencakup pengelolaan akses, enkripsi data, praktik *logging* dan *monitoring*, serta pemantauan kepatuhan terhadap regulasi dan standar yang berlaku. Dengan demikian, risiko kebocoran data atau gangguan keamanan lainnya dapat diminimalisir.

Gambar 3.3. Bukti sertifikasi *AWS Certified Solutions Architect - Associate (SAA-C03)*.

Optimasi biaya menjadi salah satu elemen yang tidak dapat diabaikan. Kandidat sertifikasi diharapkan mampu merancang solusi yang efisien dalam pemanfaatan sumber daya, mulai dari pemilihan jenis instans komputasi hingga pengaturan model pembayaran yang sesuai dengan pola penggunaan aktual. Pendekatan ini memastikan bahwa anggaran yang dialokasikan untuk kebutuhan infrastruktur dapat dimanfaatkan secara maksimal, tanpa mengorbankan kualitas atau skalabilitas layanan.

Dengan pencapaian sertifikasi *AWS Certified Solutions Architect - Associate (SAA-C03)* ini, kompetensi dalam merancang dan mengelola infrastruktur berbasis AWS menjadi lebih terarah dan berkualitas. Sertifikasi tersebut menunjukkan kesiapan untuk menghadapi tantangan nyata dalam implementasi solusi *cloud* yang andal, hemat biaya, serta tangguh menghadapi dinamika kebutuhan bisnis dan perkembangan teknologi. Dalam konteks program kerja magang, pencapaian ini juga menegaskan nilai tambah yang dapat diberikan kepada tim *Cloud & System Integration Engineer*, baik dalam meningkatkan kapabilitas teknis internal maupun dalam memberikan masukan strategis terhadap inisiatif pengembangan infrastruktur *cloud* di perusahaan.

3.4 Implementasi Infrastruktur Menggunakan Terraform di AWS

Terraform merupakan alat *Infrastructure as Code* (IaC) yang memungkinkan pengelolaan infrastruktur cloud secara deklaratif melalui file konfigurasi. Pada implementasi ini, Terraform digunakan untuk membangun infrastruktur di Amazon Web Services (AWS). Infrastruktur yang dibuat meliputi *Virtual Private Cloud (VPC)*, subnet publik, *Internet Gateway*, *Route Table*, *Security Group*, dan sebuah *instance EC2*. Proses implementasi dilakukan melalui langkah-langkah berikut:

Dalam konteks penerapan infrastruktur *cloud* yang terotomatisasi, Terraform bekerja dengan mengawali proses kerjanya melalui pendefinisian sumber daya dalam berkas konfigurasi yang ditulis menggunakan *HashiCorp Configuration Language* (HCL). Setiap elemen infrastruktur, seperti *Virtual Private Cloud* (VPC), *subnet*, atau *instance*, dideklarasikan secara deskriptif, sehingga memudahkan pengelolaan dan pemahaman. Setelah konfigurasi selesai dibuat, perintah `terraform init` dijalankan untuk mempersiapkan lingkungan kerja, sekaligus mengunduh *provider* yang relevan, misalnya *AWS Provider*, guna memastikan kesesuaian antara definisi konfigurasi dan layanan yang akan dibangun.

Tahap selanjutnya adalah perencanaan yang dilakukan menggunakan `terraform plan`. Proses ini menghasilkan rencana tindakan terperinci terhadap infrastruktur, termasuk pembuatan atau penghapusan sumber daya. Setelah memastikan rencana tersebut sesuai, perintah `terraform apply` diterapkan untuk merealisasikan konfigurasi pada lingkungan AWS. Jika kebutuhan bisnis mengalami perubahan di kemudian hari, pengguna dapat memperbarui berkas konfigurasi dan kembali menjalankan `terraform apply` agar infrastruktur dapat beradaptasi secara konsisten tanpa intervensi manual yang signifikan. Pendekatan ini sangat bermanfaat selama program kerja magang, karena memudahkan evaluasi, pengembangan, serta pemeliharaan infrastruktur *cloud*.

Proses pengembangan infrastruktur melalui Terraform dapat dilihat pada contoh konfigurasi berikut. Kode dimulai dari pendefinisian *provider* dan *region*, kemudian dilanjutkan dengan pembentukan jaringan virtual (VPC) sebagai fondasi bagi komponen lain. Setelah VPC tersedia, subnet publik, *internet gateway*, dan *route table* diatur untuk memfasilitasi komunikasi jaringan. Selanjutnya, aturan keamanan dikonfigurasikan melalui *security group*, di mana akses ke protokol tertentu diatur secara eksplisit. Pada contoh ini, digunakan *SSH (Secure Shell)* untuk koneksi aman jarak jauh ke server, *HTTP (Hypertext Transfer*

Protocol) sebagai protokol dasar untuk komunikasi web, dan *HTTPS (HTTP Secure)* yang menambahkan lapisan enkripsi sehingga pertukaran data menjadi lebih aman. Dengan demikian, pengembang dapat mengendalikan jalur lalu lintas yang diizinkan memasuki lingkungan komputasi secara terukur.

Selanjutnya, *instance EC2* diluncurkan dengan parameter yang telah ditetapkan, termasuk sistem operasi, ukuran penyimpanan, dan integrasi dengan jaringan yang baru dibuat. Fitur *output* Terraform kemudian dimanfaatkan untuk menampilkan informasi krusial, seperti alamat IP publik dari *instance*, guna memudahkan akses dan pengujian selanjutnya. Pendekatan deklaratif ini tidak hanya mempersingkat waktu pembangunan infrastruktur, tetapi juga memastikan seluruh langkah dan detail teknis terdokumentasi dengan baik. Dalam konteks program kerja magang, praktik seperti ini mendukung penerapan standar profesional, memudahkan kolaborasi dengan tim, serta mempermudah supervisi oleh pembimbing yang berpengalaman.

```
provider "aws" {
  region = "ap-southeast-3"
}
```

Kode 3.1: Konfigurasi Provider AWS dan Region

Dengan ditetapkannya provider dan region yang telah ditentukan, proses berlanjut ke penyiapan jaringan virtual. Langkah ini memungkinkan untuk membangun infrastruktur yang terisolasi, fleksibel, serta mudah diatur ulang sesuai kebutuhan dan standar perusahaan.

```
resource "aws_vpc" "main" {
  cidr_block           = "10.0.0.0/16"
  enable_dns_support   = true
  enable_dns_hostnames = true
}
```

Kode 3.2: Konfigurasi VPC

Setelah VPC terbentuk, penambahan subnet publik memberikan ruang yang terisolasi untuk sumber daya komputasi, misalnya *instance EC2*, yang memerlukan akses menuju internet atau sumber daya eksternal.

```
resource "aws_subnet" "public_subnet" {
  vpc_id            = aws_vpc.main.id
  cidr_block        = "10.0.1.0/24"
  availability_zone = "ap-southeast-3a"
}
```

Kode 3.3: Konfigurasi Subnet Publik

Langkah berikutnya adalah membangun *internet gateway* dan *route table*. Komponen ini memastikan bahwa sumber daya dalam subnet publik dapat menjangkau jaringan internet dan mendukung komunikasi data dengan sumber daya di luar lingkungan VPC.

```
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.main.id
}

resource "aws_route_table" "public_rt" {
  vpc_id = aws_vpc.main.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.igw.id
  }
}

resource "aws_route_table_association" "public_rt_assoc" {
  subnet_id      = aws_subnet.public_subnet.id
  route_table_id = aws_route_table.public_rt.id
}
```

Kode 3.4: Internet Gateway dan Route Table

Penetapan aturan keamanan melalui *security group* sangat penting untuk menjamin kendali penuh terhadap akses jaringan. Pada tahap ini, layanan *SSH*, *HTTP*, dan *HTTPS* diizinkan untuk terhubung, memberikan fleksibilitas dalam pengelolaan, pengembangan, dan pemeliharaan layanan atau aplikasi yang dijalankan pada *instance* EC2.

```
resource "aws_security_group" "awx_sg" {
  vpc_id = aws_vpc.main.id
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]  # SSH
  }
  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]  # HTTP
  }
  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]  # HTTPS
  }
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

Kode 3.5: Konfigurasi Security Group

Selanjutnya, sebuah *instance* EC2 disiapkan berdasarkan konfigurasi yang telah ditentukan. Pendekatan ini memberikan kontrol penuh atas spesifikasi *instance*, pemilihan *AMI (Amazon Machine Image)*, serta pengaturan kunci (*key pair*) untuk akses aman.

```
resource "aws_instance" "awx" {
  ami                         = "ami-04857be1146c16a59"
  instance_type               = "t3.micro"
  subnet_id                   = aws_subnet.public_subnet.id
  vpc_security_group_ids      = [aws_security_group.awx_sg.id]
  associate_public_ip_address = true

  key_name = "Testing"

  root_block_device {
    volume_size           = 20
    volume_type           = "gp3"
    delete_on_termination = true
  }

  tags = {
    Name = "Test-Provision-EC2-Terraform"
  }
}
```

Kode 3.6: Konfigurasi Instance EC2

Akhirnya, informasi-informasi penting seperti alamat IP publik *instance* dapat dikeluarkan melalui fitur *output* milik Terraform. Pendekatan ini memudahkan pengembang untuk mengetahui detail koneksi dan melakukan pengujian maupun konfigurasi lanjutan pada *instance* yang telah diluncurkan.

```
output "public_ip" {
  value = aws_instance.awx.public_ip
}
```

Kode 3.7: Output IP Publik

Dengan pendekatan deklaratif dan terstruktur ini, proses pengelolaan infrastruktur menjadi lebih efisien, mudah ditinjau, dan dapat diulang. Seluruh sumber daya yang diciptakan terdokumentasi secara sistematis, sehingga mempermudah koordinasi dengan rekan kerja atau pembimbing magang, serta memungkinkan penyesuaian yang berkelanjutan seiring dengan berkembangnya kebutuhan perusahaan dan teknologi yang digunakan.

3.4.1 Hasil Implementasi

Output dari proses provisioning menggunakan *Terraform* dapat diperiksa melalui *file state* yang dihasilkan setelah menjalankan perintah `terraform apply`. Untuk menampilkan dokumentasi lengkap dari seluruh sumber daya yang berhasil dibuat, perintah `terraform show` dapat digunakan. Perintah ini akan menampilkan konfigurasi dalam bentuk blok yang mencerminkan status terkini dari sumber daya, termasuk atribut-atribut penting seperti ID sumber daya, konfigurasi jaringan, serta parameter lainnya yang telah didefinisikan dalam file konfigurasi `.tf`. Dengan demikian, kombinasi antara *file state* dan perintah `terraform show` memungkinkan proses verifikasi hasil provisioning untuk memastikan bahwa seluruh sumber daya telah dibuat sesuai dengan spesifikasi yang diharapkan.

```
# aws_instance.awx:
resource "aws_instance" "awx" {
    ami                                  = "ami-04857be1146c16a59"
    arn                                  = "arn:aws:ec2:ap-southeast-3:354918368399:instance/i-0e02ebc7c1913c25e"
    associate_public_ip_address          = true
    availability_zone                    = "ap-southeast-3a"
    cpu_core_count                       = 1
    cpu_threads_per_core                 = 2
    disable_api_stop                     = false
    disable_api_termination              = false
    ebs_optimized                        = false
    get_password_data                    = false
    hibernation                          = false
    host_id                              = null
    iam_instance_profile                 = null
    id                                   = "i-0e02ebc7c1913c25e"
    instance_initiated_shutdown_behavior = "stop"
    instance_lifecycle                   = null
    instance_state                       = "running"
    instance_type                        = "t3.micro"
    ipv6_address_count                   = 0
    ipv6_addresses                       = []
    key_name                             = "Test-Terraform-Laporan"
    monitoring                           = false
    outpost_arn                          = null
    password_data                        = null
    placement_group                      = null
    placement_partition_number           = 0
    primary_network_interface_id         = "eni-084ba741d7f52a7e6"
    private_dns                          = "ip-10-0-1-43.ap-southeast-3.compute.internal"
    private_ip                           = "10.0.1.43"
    public_dns                           = "ec2-108-136-222-236.ap-southeast-3.compute.amazonaws.com"
    public_ip                            = "108.136.222.236"
    secondary_private_ips                = []
    security_groups                      = []
    source_dest_check                    = true
    spot_instance_request_id             = null
    subnet_id                            = "subnet-00353dccda7082239"
    tags                                 = {
        "Name" = "AWX-EC2"
    }
    tags_all                             = {
        "Name" = "AWX-EC2"
    }
    tenancy                              = "default"
    user_data_replace_on_change          = false
    vpc_security_group_ids               = [
        "sg-0bb053c98adfa93c9",
    ]

    capacity_reservation_specification {
        capacity_reservation_preference = "open"
    }
```

Gambar 3.4. *Output terraform show* untuk sumber daya *EC2 instance* (Bagian 1).

Pada gambar 3.4, terlihat bahwa sebuah *EC2 instance* telah berhasil dibuat. Atribut yang dapat diamati antara lain *AMI (Amazon Machine Image)* yang digunakan, tipe *instance* `t3.micro`, serta *Public IP* `108.136.222.236` dan *Private IP* `10.0.1.43`. Selain itu, *Key Pair* `Test-Terraform-Laporan` memastikan akses aman melalui *SSH*, sementara tag `AWX-EC2` memudahkan identifikasi *instance* tersebut dalam skala infrastruktur yang lebih luas.

```
    cpu_options {
        amd_sev_snp      = null
        core_count       = 1
        threads_per_core = 2
    }

    credit_specification {
        cpu_credits = "unlimited"
    }

    enclave_options {
        enabled = false
    }

    maintenance_options {
        auto_recovery = "default"
    }

    metadata_options {
        http_endpoint               = "enabled"
        http_protocol_ipv6          = "disabled"
        http_put_response_hop_limit = 1
        http_tokens                 = "optional"
        instance_metadata_tags      = "disabled"
    }

    private_dns_name_options {
        enable_resource_name_dns_a_record    = false
        enable_resource_name_dns_aaaa_record = false
        hostname_type                        = "ip-name"
    }

    root_block_device {
        delete_on_termination = true
        device_name           = "/dev/sda1"
        encrypted             = false
        iops                  = 3000
        kms_key_id            = null
        tags_all              = {}
        throughput            = 125
        volume_id             = "vol-0cbdbcf3884e7355d"
        volume_size           = 20
        volume_type           = "gp3"
    }
}
```

Gambar 3.5. *Output terraform show* untuk sumber daya *EC2 instance* (Bagian 2).

Bagian lanjutan ini menampilkan pengaturan teknis yang lebih mendetail, termasuk opsi CPU, skema kredit CPU (*CPU credit specification*), serta konfigurasi *root block device* yang mempengaruhi performa dan kapasitas penyimpanan *instance*. Dengan demikian, pengembang dapat memvalidasi kesesuaian spesifikasi tersebut dengan kebutuhan operasional yang telah direncanakan.

```
# aws_internet_gateway.igw:
resource "aws_internet_gateway" "igw" {
    arn      = "arn:aws:ec2:ap-southeast-3:354918368399:internet-gateway/igw-08ea409a1415ad9df"
    id       = "igw-08ea409a1415ad9df"
    owner_id = "354918368399"
    tags     = {}
    tags_all = {}
    vpc_id   = "vpc-05a1d100ec11d0389"
}

# aws_route_table.public_rt:
resource "aws_route_table" "public_rt" {
    arn              = "arn:aws:ec2:ap-southeast-3:354918368399:route-table/rtb-06e3ea047800391cf"
    id               = "rtb-06e3ea047800391cf"
    owner_id         = "354918368399"
    propagating_vgws = []
    route            = [
        {
            carrier_gateway_id         = null
            cidr_block                 = "0.0.0.0/0"
            core_network_arn           = null
            destination_prefix_list_id = null
            egress_only_gateway_id     = null
            gateway_id                 = "igw-08ea409a1415ad9df"
            ipv6_cidr_block            = null
            local_gateway_id           = null
            nat_gateway_id             = null
            network_interface_id       = null
            transit_gateway_id         = null
            vpc_endpoint_id            = null
            vpc_peering_connection_id  = null
        },
    ]
    tags             = {}
    tags_all         = {}
    vpc_id           = "vpc-05a1d100ec11d0389"
}

# aws_route_table_association.public_rt_assoc:
resource "aws_route_table_association" "public_rt_assoc" {
    gateway_id     = null
    id             = "rtbassoc-0641023c62d581b2a"
    route_table_id = "rtb-06e3ea047800391cf"
    subnet_id      = "subnet-00353dccda7082239"
}
```

Gambar 3.6. *Output terraform show* untuk *Internet Gateway* dan *Route Table*.

Pada gambar 3.6 di atas, tampak bahwa *Internet Gateway* (ID: `igw-08ea409a1415ad9df`) dan *Route Table* (ID: `rtb-06e3ea047800391cf`) telah berhasil diciptakan. Kedua komponen ini memastikan komunikasi jaringan dengan internet berjalan semestinya, di mana *Route Table* mengarahkan lalu lintas keluar (`0.0.0.0/0`) menuju *Internet Gateway*. Dengan demikian, *instance* yang berada di dalam VPC dapat terhubung ke layanan internet publik.

```
# aws_security_group.awx_sg:
resource "aws_security_group" "awx_sg" {
    arn                    = "arn:aws:ec2:ap-southeast-3:354918368399:security-group/sg-0bb053c98adfa93c9"
    description            = "Managed by Terraform"
    egress                 = [
        {
            cidr_blocks      = [
                "0.0.0.0/0",
            ]
            description      = null
            from_port        = 0
            ipv6_cidr_blocks = []
            prefix_list_ids  = []
            protocol         = "-1"
            security_groups  = []
            self             = false
            to_port          = 0
        },
    ]
    id                     = "sg-0bb053c98adfa93c9"
    ingress                = [
        {
            cidr_blocks      = [
                "0.0.0.0/0",
            ]
            description      = null
            from_port        = 22
            ipv6_cidr_blocks = []
            prefix_list_ids  = []
            protocol         = "tcp"
            security_groups  = []
            self             = false
            to_port          = 22
        },
        {
            cidr_blocks      = [
                "0.0.0.0/0",
            ]
            description      = null
            from_port        = 443
            ipv6_cidr_blocks = []
            prefix_list_ids  = []
            protocol         = "tcp"
            security_groups  = []
            self             = false
            to_port          = 443
        },
        {
            cidr_blocks      = [
                "0.0.0.0/0",
            ]
            description      = null
            from_port        = 80
            ipv6_cidr_blocks = []
            prefix_list_ids  = []
            protocol         = "tcp"
            security_groups  = []
            self             = false
            to_port          = 80
        },
    ]
```

Gambar 3.7. *Output terraform show* untuk *Security Group* (Bagian 1).


```
    name                   = "terraform-20241210032533489500000001"
    name_prefix            = "terraform-"
    owner_id               = "354918368399"
    revoke_rules_on_delete = false
    tags                   = {}
    tags_all               = {}
    vpc_id                 = "vpc-05a1d100ec11d0389"
}
```

Gambar 3.8. *Output terraform show* untuk *Security Group* (Bagian 2).

Pada gambar 3.7 dan 3.8 di atas memperlihatkan konfigurasi *Security Group* (ID: `sg-0bb053c98adfa93c9`). Aturan *ingress* yang ditetapkan mengizinkan akses melalui protokol *SSH* (port 22), *HTTP* (*port* 80), dan *HTTPS* (*port* 443) dari `0.0.0.0/0`. Hal ini menyediakan fleksibilitas bagi pengembang untuk terhubung ke *instance* guna melakukan konfigurasi awal, pengujian aplikasi web, atau manajemen lainnya. Di sisi lain, aturan *egress* mengizinkan lalu lintas keluar ke internet agar *instance* dapat memanfaatkan sumber daya eksternal, seperti pembaruan perangkat lunak atau repositori kode.

```
# aws_subnet.public_subnet:
resource "aws_subnet" "public_subnet" {
    arn                                            = "arn:aws:ec2:ap-southeast-3:354918368399:subnet/subnet-00353dccda7082239"
    assign_ipv6_address_on_creation                = false
    availability_zone                              = "ap-southeast-3a"
    availability_zone_id                           = "apse3-az1"
    cidr_block                                     = "10.0.1.0/24"
    customer_owned_ipv4_pool                       = null
    enable_dns64                                   = false
    enable_lni_at_device_index                     = 0
    enable_resource_name_dns_a_record_on_launch    = false
    enable_resource_name_dns_aaaa_record_on_launch = false
    id                                             = "subnet-00353dccda7082239"
    ipv6_cidr_block                                = null
    ipv6_cidr_block_association_id                 = null
    ipv6_native                                    = false
    map_customer_owned_ip_on_launch                = false
    map_public_ip_on_launch                        = false
    outpost_arn                                    = null
    owner_id                                       = "354918368399"
    private_dns_hostname_type_on_launch            = "ip-name"
    tags                                           = {}
    tags_all                                       = {}
    vpc_id                                         = "vpc-05a1d100ec11d0389"
}
```

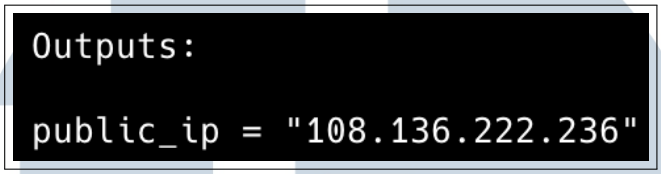
Gambar 3.9. *Output terraform show* untuk *Subnet*.

Gambar 3.9 di atas menggambarkan sebuah *Subnet Publik* (ID: `subnet-00353dccda7082239`) dengan *CIDR block* `10.0.1.0/24`. Subnet ini memberi ruang jaringan yang terisolasi namun tetap dapat diakses dari internet sesuai konfigurasi *Route Table* dan *Security Group* yang telah ditetapkan.

```
# aws_vpc.main:
resource "aws_vpc" "main" {
    arn                                  = "arn:aws:ec2:ap-southeast-3:354918368399:vpc/vpc-05a1d100ec11d0389"
    assign_generated_ipv6_cidr_block     = false
    cidr_block                           = "10.0.0.0/16"
    default_network_acl_id               = "acl-088dca2ad5cd6eb27"
    default_route_table_id               = "rtb-0dc71f8b23ec3ebf0"
    default_security_group_id            = "sg-0054246c156996e27"
    dhcp_options_id                      = "dopt-02eda06949dad7cee"
    enable_dns_hostnames                 = true
    enable_dns_support                   = true
    enable_network_address_usage_metrics = false
    id                                   = "vpc-05a1d100ec11d0389"
    instance_tenancy                     = "default"
    ipv6_association_id                  = null
    ipv6_cidr_block                      = null
    ipv6_cidr_block_network_border_group = null
    ipv6_ipam_pool_id                    = null
    ipv6_netmask_length                  = 0
    main_route_table_id                  = "rtb-0dc71f8b23ec3ebf0"
    owner_id                             = "354918368399"
    tags                                 = {}
    tags_all                             = {}
}
```

Gambar 3.10. *Output terraform show* untuk *VPC*.

Pada gambar 3.10 tampak *VPC* (ID: `vpc-05a1d100ec11d0389`) dengan *CIDR block* `10.0.0.0/16`. VPC ini berperan sebagai pondasi jaringan privat di dalam AWS, memfasilitasi kontrol penuh atas topologi, konfigurasi, dan pengamanan infrastruktur jaringan yang digunakan.



Gambar 3.11. *Output terraform show* untuk informasi *public_ip* yang dihasilkan oleh konfigurasi.

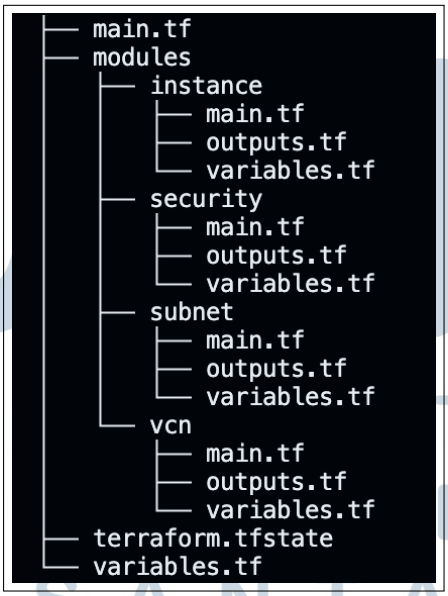
Gambar 3.11 ini menampilkan informasi `public_ip = "108.136.222.236"` yang merupakan hasil dari blok *output* dalam konfigurasi Terraform. Informasi ini mempermudah pengembang untuk langsung mengakses *instance* yang telah diprovisikan, baik untuk pengujian aplikasi, pengaturan tambahan, maupun proses pemeliharaan berkelanjutan.

Dengan tersedianya tangkapan layar ini, proses verifikasi dan dokumentasi hasil *provisioning* menjadi lebih transparan dan mudah. Pengguna dapat membandingkan keluaran aktual dengan konfigurasi yang telah ditulis, memastikan bahwa setiap komponen infrastruktur, seperti *VPC*, *Subnet*, *Internet Gateway*, *Route Table*, *Security Group*, serta *EC2 instance* telah berhasil diimplementasikan sesuai rencana awal. Dalam konteks magang, dokumentasi terperinci ini memudahkan proses peninjauan oleh pembimbing dan menjadi bukti konkret kemampuan dalam mengelola infrastruktur *cloud* menggunakan Terraform.

3.5 Implementasi Infrastruktur Menggunakan Terraform di Oracle Cloud Infrastructure (OCI)

Pada implementasi di lingkungan OCI, pendekatan *modular* diterapkan untuk memudahkan pengembangan, pemeliharaan, serta penyesuaian komponen infrastruktur secara berkelanjutan. Struktur direktori di bawah ini menunjukkan pembagian konfigurasi ke dalam beberapa modul terpisah, masing-masing bertanggung jawab terhadap satu aspek infrastruktur seperti `VCN`, `security list`, `subnet`, dan `instance`. Pendekatan ini memudahkan pengelolaan skala besar, memungkinkan perubahan dilakukan secara terukur tanpa mempengaruhi komponen lain.

Dalam konteks penerapan infrastruktur *cloud* yang terotomatisasi, Terraform bekerja dengan mengawali proses kerjanya melalui pendefinisian sumber daya dalam berkas konfigurasi yang ditulis menggunakan *HashiCorp Configuration Language* (HCL). Setiap elemen infrastruktur dideklarasikan secara deskriptif, sehingga memudahkan pengelolaan dan pemahaman. Setelah konfigurasi selesai dibuat, perintah `terraform init` dijalankan untuk mempersiapkan lingkungan kerja, sekaligus mengunduh *provider* yang relevan, misalnya OCI Provider, guna memastikan kesesuaian antara definisi konfigurasi dan layanan yang akan dibangun.



Gambar 3.12. *Tree view* dari struktur konfigurasi Terraform yang diimplementasikan

Struktur direktori yang ditampilkan pada gambar 3.12 terdiri dari file utama `main.tf` yang mengatur pemanggilan modul, serta file `variables.tf` dan

`outputs.tf` yang mendukung konfigurasi variabel dan output.

Konfigurasi dimulai dengan mendefinisikan provider OCI pada file `main.tf` di root direktori. File ini juga memanggil beberapa modul yang berfungsi untuk membangun `VCN`, `security list`, `subnet`, dan `instance`.

```
provider "oci" {
  region = "us-chicago-1"
}

module "vcn" {
  source         = "./modules/vcn"
  compartment_id = var.compartment_ocid
}

module "security" {
  source         = "./modules/security"
  compartment_id = var.compartment_ocid
  vcn_id         = module.vcn.vcn_id
}

module "subnet" {
  source              = "./modules/subnet"
  compartment_id      = var.compartment_ocid
  availability_domain = "iGio:US-CHICAGO-1-AD-1"
  vcn_id              = module.vcn.vcn_id
  route_table_id      = module.vcn.route_table_id
  dhcp_options_id     = module.vcn.dhcp_options_id
  security_list_id    = module.security.security_list_id
}

module "instance" {
  source             = "./modules/instance"
  compartment_id     = var.compartment_ocid
  availability_domain = "iGio:US-CHICAGO-1-AD-1"
  subnet_id          = module.subnet.subnet_id
  custom_image_ocid  = var.custom_image_ocid
}

output "instance_public_ip" {
  value = module.instance.instance_public_ip
}
```

Kode 3.8: Root Module - main.tf

Kode 3.8 dimulai dengan mendefinisikan provider OCI, yang menentukan region di mana infrastruktur akan dikelola. Modul-modul yang dipanggil meliputi `vcn`, `security`, `subnet`, dan `instance`. Modul `vcn` akan membuat *Virtual Cloud Network*, modul `security` akan membuat *security list*, modul `subnet` akan mengatur pembuatan *subnet*, dan modul `instance` akan membuat *instance compute* di dalam *subnet* yang telah dibuat. Setiap modul menerima variabel seperti `compartment_ocid` dan `custom_image_ocid` sebagai input untuk mengkonfigurasi sumber daya dengan benar.

File variabel yang digunakan pada *root module* adalah sebagai berikut:

```
variable "compartment_ocid" {
  type        = string
  description = "Compartment OCID for OCI resources"
  default     = "ocid1.compartment.oc1..-##########"
}

variable "custom_image_ocid" {
  type        = string
  description = "OCID for Custom RHEL 7.7 image in OCI"
  default     = "ocid1.image.oc1.us-chicago-##########"
}
```

Kode 3.9: Root Module - variables.tf

Pada kode 3.9, dua variabel didefinisikan: `compartment_ocid` yang menyimpan OCID untuk compartment OCI dan `custom_image_ocid` yang menyimpan OCID untuk image kustom yang digunakan dalam pembuatan *instance*. Variabel ini akan digunakan dalam file `main.tf` untuk mengonfigurasi sumber daya di OCI.

Modul Instance bertanggung jawab untuk pembuatan *instance* di dalam subnet yang telah dikonfigurasi sebelumnya.

```
resource "oci_core_instance" "custom_instance" {
  compartment_id      = var.compartment_id
  availability_domain = var.availability_domain
  display_name        = "Ansible_Tower_OCI_Core_LandingZone"
  shape               = "VM.Standard.E5.Flex"
  shape_config {
    ocpus         = 2
    memory_in_gbs = 16
  }
  create_vnic_details {
    subnet_id        = var.subnet_id
    assign_public_ip = true
  }
  source_details {
    source_type = "image"
    source_id   = var.custom_image_ocid
  }
  metadata = {
    ssh_authorized_keys = file("${path.module}/Terraform.pub")
  }
}
```

Kode 3.10: Instance Module - main.tf

Modul ini mendefinisikan sumber daya `oci_core_instance` yang membuat *instance* di OCI. *Instance* tersebut diatur agar menggunakan dua vCPU dan 16 GB memori dengan shape `VM.Standard.E5.Flex`. IP publik akan diberikan kepada *instance* tersebut. Gambar kustom yang akan digunakan berasal dari `custom_image_ocid`. Selain itu, metadata *instance* juga disiapkan untuk mengotorisasi kunci SSH yang disimpan di file `Terraform.pub`.

```
variable "compartment_id" {
  description = "Compartment OCID for the Instance"
  type        = string
}
variable "availability_domain" {
  description = "Availability Domain for the Instance"
  type        = string
}
variable "subnet_id" {
  description = "Subnet ID where the Instance will be created"
  type        = string
}
variable "custom_image_ocid" {
  description = "Custom Image OCID for the Instance"
  type        = string
}
```

Kode 3.11: Instance Module - variables.tf

Kode 3.11 mendefinisikan variabel-variabel yang digunakan oleh modul *instance*, seperti `compartment_id`, `availability_domain`, `subnet_id`, dan `custom_image_ocid`. Nilai-nilai variabel ini akan diberikan melalui input saat menjalankan Terraform dan digunakan untuk mengonfigurasi *instance* dengan benar.

```
output "instance_public_ip" {
  value = oci_core_instance.custom_instance.public_ip
}
```

Kode 3.12: Instance Module - outputs.tf

Kode 3.12 ini mendefinisikan *output* `instance_public_ip` yang mengembalikan alamat IP publik dari *instance* yang telah dibuat. *Output* ini memudahkan untuk mengambil informasi penting tentang sumber daya yang telah dikelola oleh Terraform.

Modul Security bertanggung jawab untuk mengatur *security list* yang mendefinisikan aturan lalu lintas jaringan.

```
resource "oci_core_security_list" "allow_all_sg" {
  compartment_id = var.compartment_id
  vcn_id         = var.vcn_id
  display_name   = "Allow_All_Security_List"
  ingress_security_rules {
    protocol = "6"  # TCP
    source   = "0.0.0.0/0"
    tcp_options {
      min = 22
      max = 22
    }
  }
  egress_security_rules {
    protocol    = "all"
    destination = "0.0.0.0/0"
  }
}
```

Kode 3.13: Security Module - main.tf

Modul ini membuat *security list* yang mengatur akses jaringan ke dan dari *instance*. Aturan `ingress` memungkinkan akses TCP pada *port* 22 (SSH) dari semua alamat IP, sementara aturan `egress` mengizinkan semua jenis lalu lintas keluar menuju semua alamat IP. Hal ini memungkinkan komunikasi dengan *instance* yang dibuat di OCI.

```
variable "compartment_id" {
  description = "Compartment OCID for security list"
  type        = string
}

variable "vcn_id" {
  description = "VCN ID where security list is applied"
  type        = string
}
```

Kode 3.14: Security Module - variables.tf

Kode 3.14 mendefinisikan variabel `compartment_id` yang digunakan untuk menunjuk ke compartment tempat security list diterapkan, serta `vcn_id` yang menunjuk ke *Virtual Cloud Network* tempat *security list* ini akan dihubungkan.

```
output "security_list_id" {
  value = oci_core_security_list.allow_all_sg.id
}
```

Kode 3.15: Security Module - outputs.tf

Kode 3.15 mendefinisikan *output* `security_list_id`, yang mengembalikan ID dari security list yang telah dibuat. *Output* ini digunakan untuk referensi pada modul-modul lain yang membutuhkan ID *security list* ini, seperti

```
resource "oci_core_subnet" "public_subnet" {
  compartment_id             = var.compartment_id
  availability_domain        = var.availability_domain
  display_name               = "Public_Subnet"
  vcn_id                     = var.vcn_id
  cidr_block                 = "10.10.1.0/24"
  route_table_id             = var.route_table_id
  dhcp_options_id            = var.dhcp_options_id
  security_list_ids          = [var.security_list_id]
  prohibit_public_ip_on_vnic = false
}
```

Kode 3.16: Subnet Module - main.tf

Modul Subnet pada kode 3.16 mendefinisikan sumber daya `oci_core_subnet` yang bertugas membuat subnet publik di dalam VCN yang telah ditentukan. Subnet memiliki blok CIDR `10.10.1.0/24` dan terhubung dengan `route_table_id`, `dhcp_options_id`, dan `security_list_id` yang telah dibuat sebelumnya. Konfigurasi ini mengizinkan penggunaan IP publik pada *VNIC*.

```
variable "compartment_id" {
  description = "Compartment OCID for Subnet"
  type        = string
}

variable "availability_domain" {
  description = "Availability Domain for the Subnet"
  type        = string
}

variable "vcn_id" {
  description = "VCN ID where the Subnet will be created"
  type        = string
}

variable "route_table_id" {
  description = "Route Table ID for the Subnet"
  type        = string
}

variable "dhcp_options_id" {
  description = "DHCP Options ID for the Subnet"
  type        = string
}

variable "security_list_id" {
  description = "Security List ID for the Subnet"
  type        = string
}
```

Kode 3.17: Subnet Module - variables.tf

Kode 3.17 mendefinisikan variabel-variabel yang digunakan untuk membuat subnet. Variabel-variabel ini meliputi `compartment_id`, `availability_domain`, `vcn_id`, `route_table_id`, `dhcp_options_id`, dan `security_list_id`. Nilai-nilai variabel ini diberikan saat menjalankan Terraform.

```
output "subnet_id" {
  value = oci_core_subnet.public_subnet.id
}
```

Kode 3.18: Subnet Module - outputs.tf

Kode 3.18 mendefinisikan *output* `subnet_id` yang mengembalikan ID subnet yang telah dibuat. *Output* ini mempermudah modul lain untuk menggunakan subnet sebagai referensi dengan cara menyediakan ID yang diperlukan untuk menghubungkan sumber daya lainnya, seperti *instances* atau komponen jaringan lainnya, ke subnet tersebut. Dengan mendeklarasikan *output* ini, komunikasi antar-modul menjadi lebih efisien dan mengurangi potensi kesalahan konfigurasi yang disebabkan oleh pengisian nilai secara manual. *Output* ini juga mendukung otomatisasi penuh dalam membangun infrastruktur yang kompleks di dalam OCI.

```
resource "oci_core_vcn" "primary_vcn" {
  compartment_id = var.compartment_id
  display_name   = "Primary_VCN"
  cidr_block     = "10.10.0.0/16"
  dns_label      = "primaryvcn"
}

resource "oci_core_internet_gateway" "internet_gateway" {
  compartment_id = var.compartment_id
  vcn_id         = oci_core_vcn.primary_vcn.id
  display_name   = "Internet_Gateway"
}

resource "oci_core_route_table" "internet_route" {
  compartment_id = var.compartment_id
  vcn_id         = oci_core_vcn.primary_vcn.id
  display_name   = "Internet_Route_Table"

  route_rules {
    destination       = "0.0.0.0/0"
    network_entity_id = oci_core_internet_gateway.internet_gateway
    .id
  }
}

resource "oci_core_dhcp_options" "custom_dhcp" {
  compartment_id = var.compartment_id
  vcn_id         = oci_core_vcn.primary_vcn.id
  display_name   = "Custom_DHCP_Options"

  options {
    type        = "DomainNameServer"
    server_type = "VcnLocalPlusInternet"
  }

  options {
    type                = "SearchDomain"
    search_domain_names = ["primaryvcn.oraclevcn.com"]
  }
}
```

Kode 3.19: VCN Module - main.tf

Modul VCN mendefinisikan sumber daya `oci_core_vcn` yang

bertugas membuat *Virtual Cloud Network* dengan blok CIDR `10.10.0.0/16`. `oci_core_internet_gateway` digunakan untuk mengatur akses internet ke VCN, sedangkan `oci_core_route_table` mengelola aturan rute yang mengarahkan lalu lintas internet melalui *Internet Gateway*. Selain itu, `oci_core_dhcp_options` mengonfigurasi DHCP dengan server DNS dan domain pencarian kustom untuk VCN.

```
variable "compartment_id" {
  description = "Compartment OCID for VCN"
  type        = string
}
```

Kode 3.20: VCN Module - variables.tf

Kode 3.20 mendefinisikan variabel `compartment_id` yang digunakan untuk menunjuk ke *compartment* tempat VCN dan sumber daya terkait akan dibuat.

```
output "vcn_id" {
  value = oci_core_vcn.primary_vcn.id
}

output "route_table_id" {
  value = oci_core_route_table.internet_route.id
}

output "dhcp_options_id" {
  value = oci_core_dhcp_options.custom_dhcp.id
}
```

Kode 3.21: VCN Module - outputs.tf

Kode 3.21 mendefinisikan tiga *output*: `vcn_id` yang mengembalikan ID dari VCN yang telah dibuat, `route_table_id` yang mengembalikan ID tabel rute internet, dan `dhcp_options_id` yang mengembalikan ID opsi DHCP kustom. *Output* ini digunakan oleh modul lain yang membutuhkan referensi ke sumber daya yang dibuat di dalam VCN.

Dengan pendekatan *modular* ini, semua konfigurasi infrastruktur OCI ditangani secara terpisah, mempermudah pengelolaan dan pemeliharaan sumber daya. Setiap modul berfungsi secara independen namun tetap saling terhubung melalui parameter input dan *output*.

3.6 Kendala dan Solusi yang Ditemukan

Selama masa kerja magang di PT InfraCom Technology, sejumlah kendala muncul dan memerlukan pendekatan sistematis untuk penyelesaiannya. Tantangan tersebut berkaitan dengan penguasaan pengetahuan dasar serta aspek teknis yang kompleks. Di bawah ini diuraikan kendala-kendala yang dihadapi beserta solusi yang diterapkan.

Beberapa kendala yang ditemui dalam proses implementasi adalah sebagai berikut:

1. Terbatasnya pengetahuan dasar mengenai *IT Enterprise Asset Management*, infrastruktur TI, serta teknologi *cloud computing*. Kondisi ini mengakibatkan kesulitan dalam memahami konteks penerapan teknologi tersebut di lingkungan industri yang sarat kompleksitas.
2. Kendala teknis pada proses implementasi Terraform di Ansible Tower. Integrasi ini memerlukan *debugging* mendalam, terutama karena adanya sistem isolasi *bubblewrap* pada lingkungan Linux yang memperumit eksekusi *source code* Terraform.

Untuk mengatasi kendala-kendala di atas, solusi berikut diterapkan:

1. Melakukan eksplorasi dan pembelajaran mandiri terkait *IT Enterprise Asset Management* serta *cloud computing* melalui dokumentasi resmi, artikel teknis, video pembelajaran, dan platform kursus daring. Pendekatan ini memfasilitasi penguasaan konsep dasar yang menjadi fondasi kritis dalam memahami dan menerapkan teknologi tersebut pada proyek.
2. Melakukan konsultasi dengan pihak yang lebih berpengalaman dan melakukan eksperimen lanjutan terkait implementasi Terraform di Ansible Tower. Upaya ini mencakup pengujian ulang, penyesuaian konfigurasi, dan strategi *debugging* yang lebih efektif untuk mengatasi isolasi *bubblewrap*, sehingga integrasi berjalan sesuai harapan.