Book Library Documentation

Anggota :

1. Dzaky Purnomo Rifa'i (5026221085)
2. Darrell Valentino (5026221086)
3. Frans Nicklaus Gusyanto (5026221089)
4. Viera Tito Virgiawan (5026221096)

Deskripsi

Book Library adalah sebuah app untuk menyimpan daftar buku-buku apa saja yang tersedia dalam toko. Daftar buku dilengkapi dengan nama buku, author, harga, deskripsi, serta gambar. App ini memperbolehkan user untuk melihat buku apa saja yang ada dalam sebuah toko buku.

Dalam proyek ini, kelompok kami berfokus pada penerapan alat DevOps untuk proses build dan deployment aplikasi serta pemantauan aktivitas di dalamnya. Isi dari dokumentasi ini akan mencakup tentang proses pengerjaan proyek pembuatan Todo-List Application kelompok kami, jenis-jenis *framework* dan *tools* yang digunakan di dalam proyek dan *user-flow* dari pengguna itu sendiri.

Link repository :
https://github.com/dzaky-pr/fp-pso

Tools dan software yang digunakan

Dalam project ini kami menggunakan beragam tools dengan kegunaannya masing-masing dalam seluruh tahapan DevOps yang mencakup berbagai macam hal, dari code checker, version control, CI/CD, deployment, security scanning serta monitoring. Berikut adalah detail tools dan software yang kami gunakan :

1. Integrated Development Environment (IDE) : VSCode
2. Framework : Next.js 14
3. Version Control : Git + Github
4. Infrastrucure Environment : Amazon Lambda, DynamoDB, EC2, API Gateway, S3, IAM

5. CI/CD Pipeline : Github Actions
6. CI tools : Husky, Jest, Biome
7. CD tools : Playwright
8. Security Scanning : SonarCloud
9. Monitoring : Amazon CloudWatch, Amazon SNS

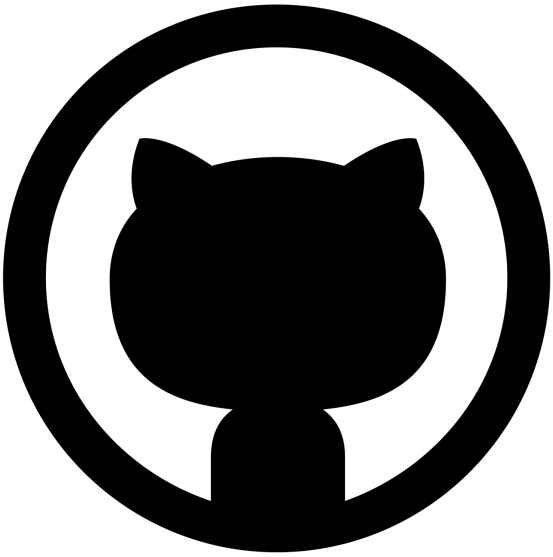
VSCode (Visual Studio Code)

Visual Studio Code adalah *Integrated Development Environment (IDE)* ringan namun powerful yang kami gunakan dalam pengembangan proyek. VSCode mendukung berbagai bahasa pemrograman dan dilengkapi dengan fitur seperti IntelliSense, debugging, dan integrasi Git. Selain itu, kami memanfaatkan berbagai ekstensi seperti Tailwind CSS IntelliSense, ESLint, dan Prettier untuk meningkatkan produktivitas dan menjaga konsistensi kode selama pengembangan.

Next.js 14

Next.js 14 adalah framework berbasis React yang kami gunakan untuk membangun aplikasi web fullstack. Framework ini menyediakan fitur built-in seperti server-side rendering (SSR), static site generation (SSG), App Router, API routes, dan middleware yang mempermudah pengembangan aplikasi modern. Dengan Next.js, kami dapat mengoptimalkan performa, SEO, dan manajemen routing secara efisien.

GitHub



Git adalah sistem version control yang digunakan untuk melacak perubahan kode secara efisien, sedangkan GitHub adalah platform berbasis web tempat kami menyimpan dan mengelola repositori proyek. Selain menyimpan kode, kami juga memanfaatkan GitHub Project untuk manajemen tugas, dengan menggunakan template Kanban board. Kanban board ini memvisualisasikan alur kerja mulai dari *To Do*, *In Progress*, hingga *Done*, sehingga tim dapat memantau progres setiap fitur secara transparan dan terorganisir.

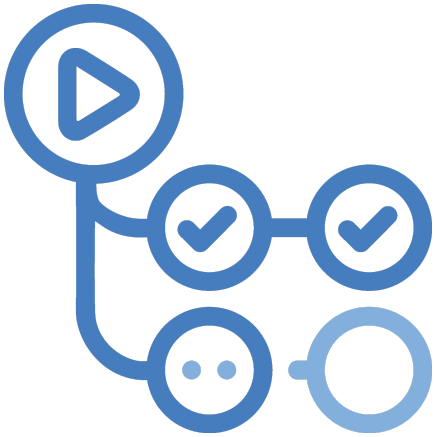
AWS (Amazon Web Service)

Untuk infrastruktur, kami menggunakan beberapa layanan dari AWS:

- **Amazon Lambda** untuk menjalankan backend secara serverless.
- **DynamoDB** sebagai database NoSQL cepat dan skalabel.
- **EC2** sebagai server virtual yang digunakan untuk tugas-tugas komputasi tertentu.
- **API Gateway** sebagai penghubung HTTP ke Lambda atau backend.
- **S3** untuk penyimpanan objek seperti gambar, file statis, atau log.
- **IAM** (Identity and Access Management) untuk mengatur hak akses antar layanan agar tetap aman.

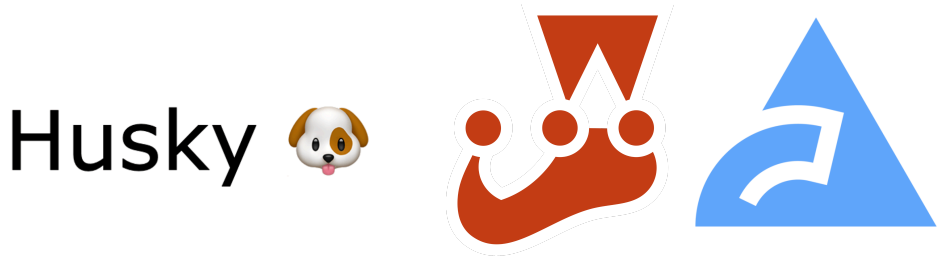
Dengan memanfaatkan layanan ini, kami dapat membangun aplikasi berbasis cloud yang fleksibel, hemat biaya, dan mudah diskalakan.

GitHub Actions



GitHub Actions adalah alat otomasi yang kami gunakan untuk mengatur pipeline Continuous Integration / Continuous Delivery (CI/CD). Setiap kali ada push atau pull request ke repositori, GitHub Actions akan menjalankan serangkaian job otomatis, seperti linting, unit testing, build, hingga deploy ke lingkungan produksi. Dengan pendekatan ini, kami memastikan bahwa setiap kode yang digabung ke main branch telah melewati proses verifikasi otomatis yang ketat.

Husky, Jest, dan Biome



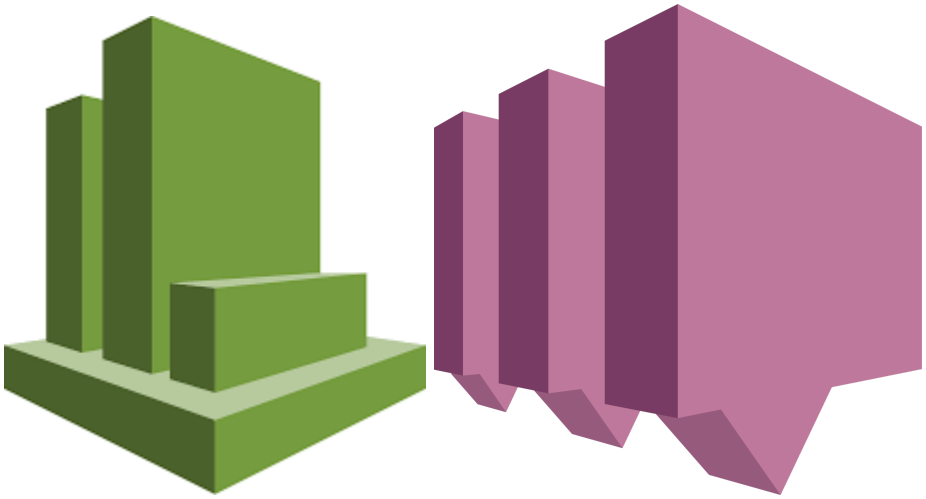
- **Husky** kami gunakan untuk menjalankan *pre-commit* dan *pre-push hooks*. Misalnya, sebelum commit diproses, Husky akan menjalankan lint dan unit test secara otomatis agar standar kualitas selalu terjaga.
- **Jest** digunakan sebagai framework unit testing utama untuk menguji fungsionalitas aplikasi, baik pada level logika maupun komponen.
- **Biome** adalah tool modern linting dan formatting (pengganti ESLint dan Prettier) yang menjaga konsistensi gaya penulisan kode di seluruh tim.

Dengan kombinasi ketiganya, kami membangun fondasi kuat untuk maintainability dan reliability dari kode yang dikembangkan.

Playwright

Playwright adalah alat *end-to-end testing* yang kami gunakan untuk menguji interaksi pengguna secara otomatis, seperti klik tombol, pengisian form, dan navigasi antar halaman. Dengan Playwright, kami dapat mensimulasikan perilaku pengguna nyata di berbagai browser dan memastikan semua fitur berjalan dengan baik sebelum dideploy ke produksi.

Amazon CloudWatch dan Amazon SNS



Amazon CloudWatch kami gunakan untuk memantau performa sistem dan mengumpulkan log dari layanan seperti Lambda dan EC2. Kami menetapkan alarm pada metrik penting seperti error rate atau latency.

Jika terjadi anomali, **Amazon SNS (Simple Notification Service)** akan mengirim notifikasi otomatis ke tim melalui email atau channel tertentu.

Set Up Tools

Dalam bagian ini akan dijelaskan cara inisialisasi tools-tools yang akan digunakan untuk menjalankan DevOps kelompok kami.

Github

1. Login ataupun register akun Github pada website GitHub
2. Bisa masuk ke halaman profile akun yang sedang dipakai lalu membuat rapository baru, set menjadi public dan masukkan nama yang sesuai. Pada kasus kami, yaitu "fp-pso"
3. Menambahkan collaborator yang akan bekerja pada repo tersebut dengan mengscroll ke bawah dan di bagian kanan ada collaborator. Dapat mengadd colaborator dengan memasukkan username github akun orang tersebut dan jangan lupa untuk memverifikasi melalui email invitationnya.
4. Github ini akan kami pakai sebagai dashboard untuk version control Git serta juga sebagai cara mudah melihat CI CD pipeline yang akan kami gunakan melalui Github Actions.

AWS CLI

1. Menginstall AWS Cli sesuai dengan operating system yang akan digunakan. Berikut adalah link untuk Windows 64-bit (https://awscli.amazonaws.com/AWSCLIV2.msi)
2. Lalu klik .msi file nya lalu klik next-next lalu install untuk install AWS Cli.
3. Untuk mengecek sudah terinstall atau belum, buka command prompt dan masukkan command ini `aws --version` Dan seharusnya mengeluarkan ini.

```
C:\Users\Frans>aws --version
aws-cli/2.27.33 Python/3.13.3 Windows/11 exe/AMD64
```

Node.js

1. Untuk menginstall Node.js maka buka website https://nodejs.org dan klik Download Node.js (LTS).
2. Jalankan installernya dan tunggu hingga finish.
3. Untuk mengecek sudah terdownload, maka coba run command `node -v` serta `npm -v` .Maka seharusnya mengeluarkan output seperti ini.

```
PS C:\Users\Frans> node -v
v22.16.0
PS C:\Users\Frans> npm -v
10.9.2
```

Terraform

1. Buka website Install | Terraform | HashiCorp Developer, pilih operating system yang sesuai, dalam kasus kami adalah Windows makan pilih yang Windows AMD64.
2. Extract ZIP nya lalu letakkan terraform.exe dalam path yang akan digunakan
3. Untuk mengecek sudah terdownload, maka coba run `terraform version` hasilnya adalah sebagai berikut.

```
PS C:\Users\Frans> terraform version
Terraform v1.13.0-alpha20250521
on windows_amd64
```

Amazon Web Service

1. Buka website Cloud Computing Services - Amazon Web Services (AWS) lalu buat akun Root untuk membuka aws console.
2. Pastikan payment info yang dimasukkan dapat terverifikasi, dapat menggunakan Visa ataupun MasterCard. Agar dapat menggunakan free trial 12 month dari AWS.
3. Untuk services yang akan digunakan akan dijelaskan step by step pada bagian selanjutnya secara jelas.

Node Packages

1. Husky : `npm install husky`
2. Biome : `npm install biome`
3. Playwright : `npm install playwright`
4. Jest : `npm install jest`

Set Up environment

Dalam bagian ini akan ada dokumentasi step by step cara set up environment yang akan digunakan untuk infrastruktur web. Berikut merupakan langkah-langkah yang komprehensif dan bertahap pada kegiatan inisialisasi environment project Bookstore.

Set Up Book Library (Initial Repo) :

1. Pertama, kami melakukan fork repository repo Book Library milik TheTechMaze ke akun GitHub kami. Kunjungi link https://github.com/thetechmaze/book-library-next-dynamo dan klik "Fork".
2. Setelah repository tersebut berhasil difork, clone repo ke mesin lokal kami.
3. Masuk ke folder proyek dan install dependencies yang diperlukan. Dengan cara cd fp-pso dan diikuti dengan npm install.
4. Kemudian kami melakukan Set Up AWS Services dengan langkah-langkah sebagai berikut:

   AWS DynamoDB:

   a. Masuk ke AWS Management Console.
   b. Akses layanan DynamoDB dan klik Create Table
   c. Memasukkan nama tabel (BooksLibrary) dan mendefinisikan partition key, dalam hal ini kami menggunakan id (string) sebagai id dari buku.
   d. Setelah itu kami mengatur pengaturan tabel sesuai kebutuhan, dan klik Create.

   AWS Lambda:

   a. Masuk ke AWS Management Console.
   b. Pilih layanan Lambda dan klik Create Function.
   c. Pilih Author from Scratch, beri nama fungsi dalam hal ini BookLambda.
   d. Pilih Node.js 18.0.20 sebagai runtime.
   e. Masukkan kode JavaScript untuk menangani API.
   f. Tentukan role IAM yang diperlukan untuk akses DynamoDB.
   g. Klik Create Function.

   AWS API Gateway:

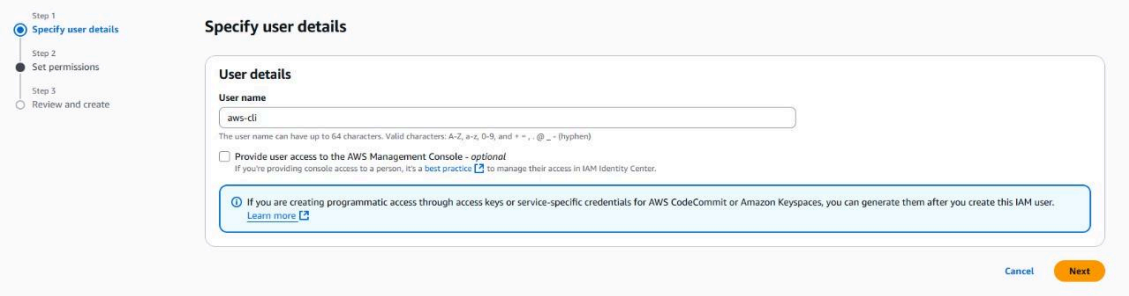
   a. Masuk ke AWS Management Console.
   b. Pilih API Gateway dan klik Create API.
   c. Pilih HTTP API untuk API yang lebih sederhana atau REST API untuk API yang lebih kompleks.

d. Menentukan endpoint, dalam hal ini /books dan /books[id] untuk detail buku yang dipilih, dan sambungkan dengan Lambda function yang telah dibuat.
e. Terapkan pengaturan API Gateway dan ambil URL endpoint.

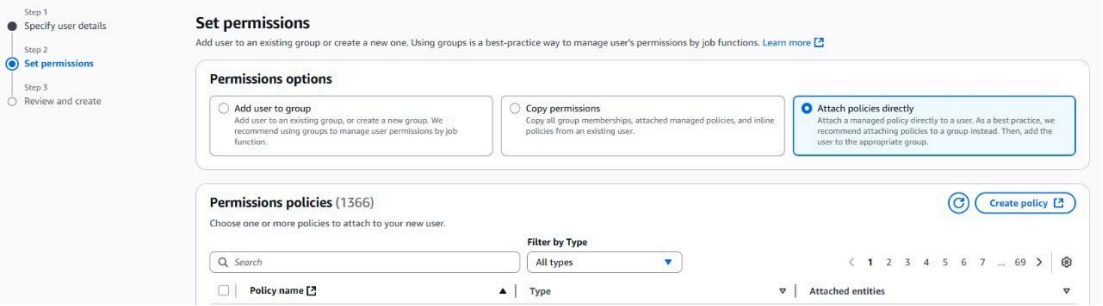
5. Setelah menyiapkan AWS services, buat file .env.local di root directory proyek dan tambahkan variabel berikut:
   AWS_API_URL=https://your-api-id.execute-api.your-region.amazonaws.com/dev
   Pastikan untuk mengganti your-api-id dengan ID API Gateway yang sesuai.
6. Setelah semua pengaturan dilakukan jalankan server pengembangan menggunakan perintah npm run dev, lalu buka http://localhost:3000 di browser untuk melihat aplikasi berjalan secara lokal.

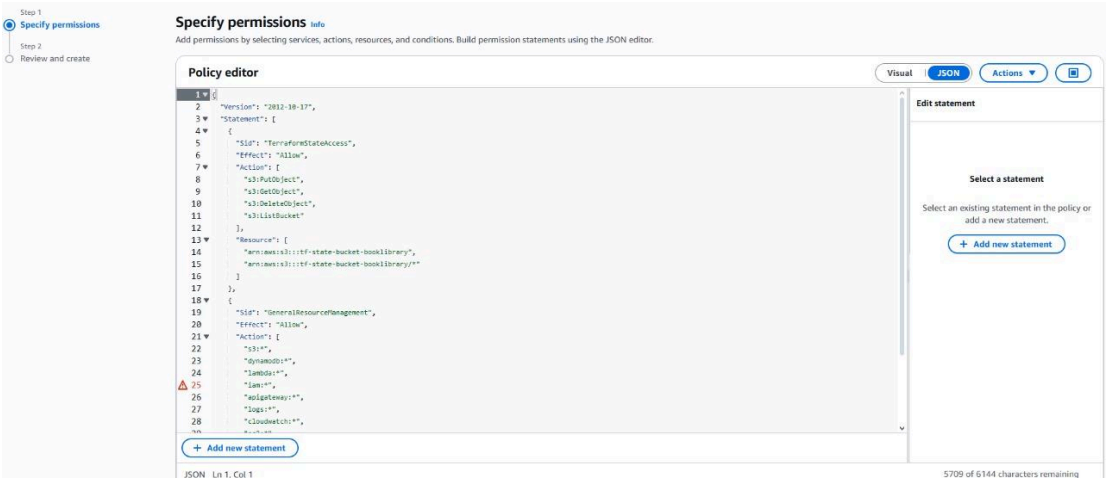
Set Up IAM User for AWS CLI (Used for Terraform Credentials) :

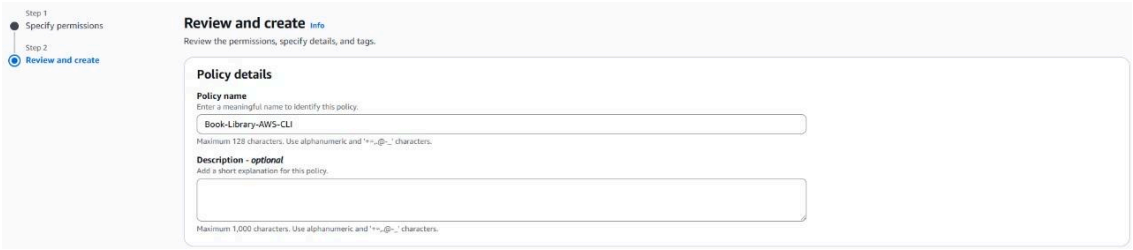
1. Pertama, buka AWS Console dan masuk ke bagian IAM, lalu klik **Buat Pengguna** untuk membuat pengguna IAM baru.



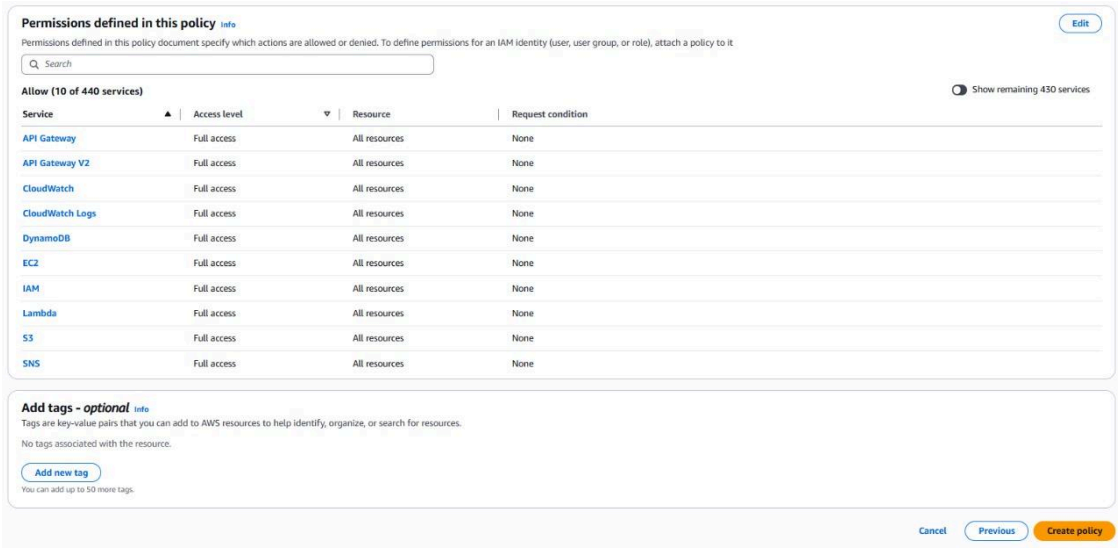
2. Selanjutnya, pada bagian izin, pilih Lampirkan kebijakan langsung dan klik Buat kebijakan. Pilih JSON dan salin konten dari ./terraform/iam-policy-template.txt, kemudian tempelkan ke editor teks di AWS Console. Beri nama untuk kebijakan tersebut dan klik Buat.



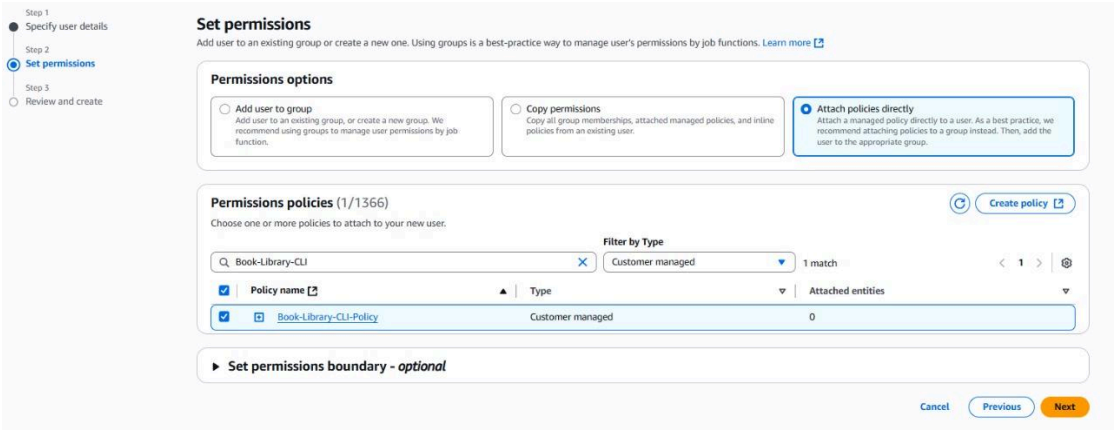


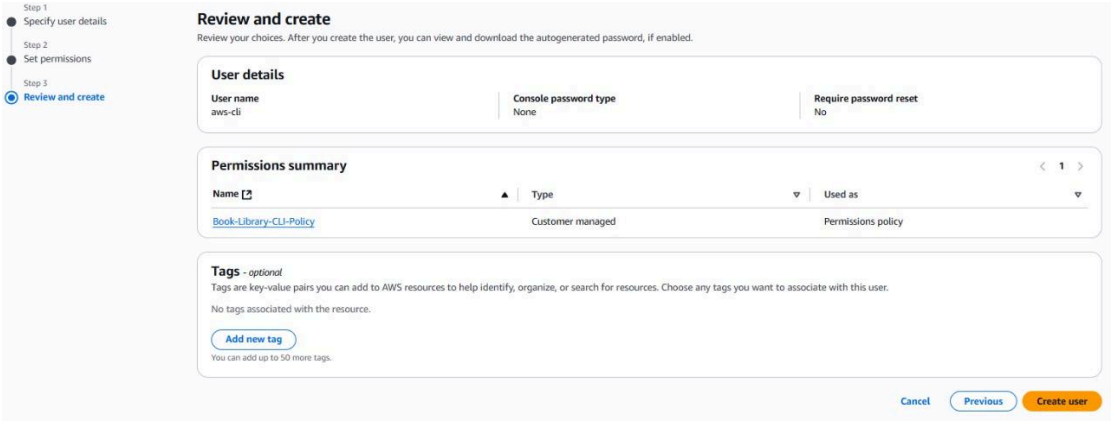


3. Kembali ke halaman pembuatan pengguna IAM, cari kebijakan yang baru saja dibuat (gunakan filter Customer Managed untuk mempermudah pencarian), kemudian lampirkan kebijakan tersebut ke pengguna yang baru dibuat.

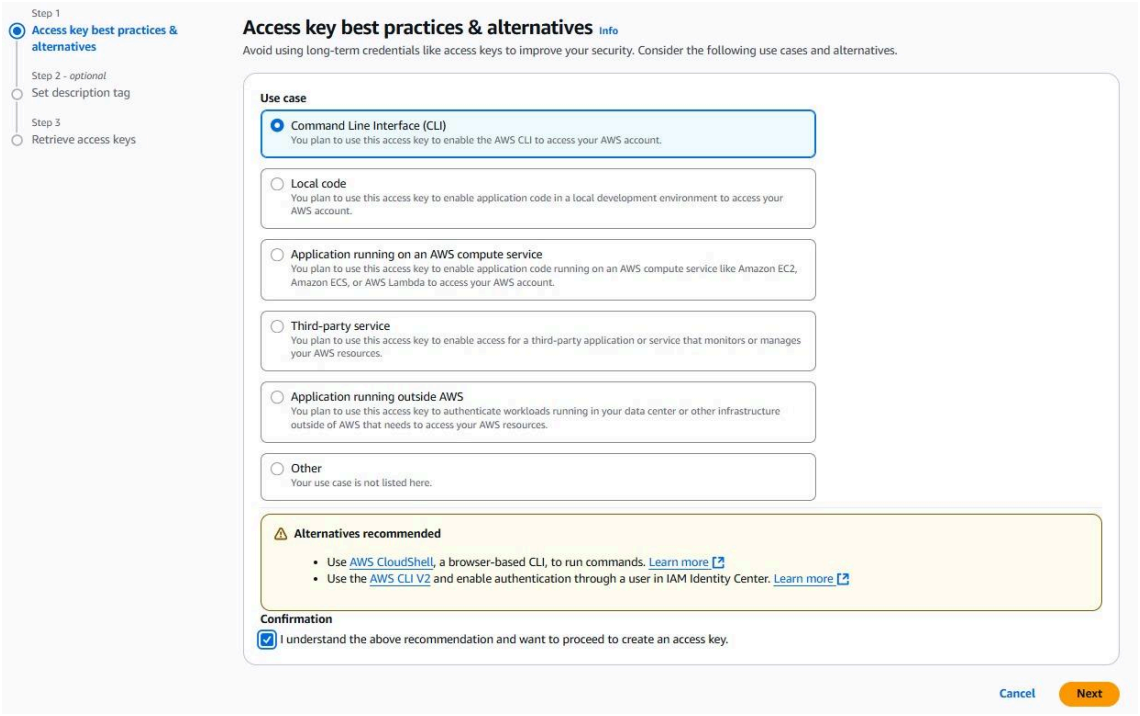


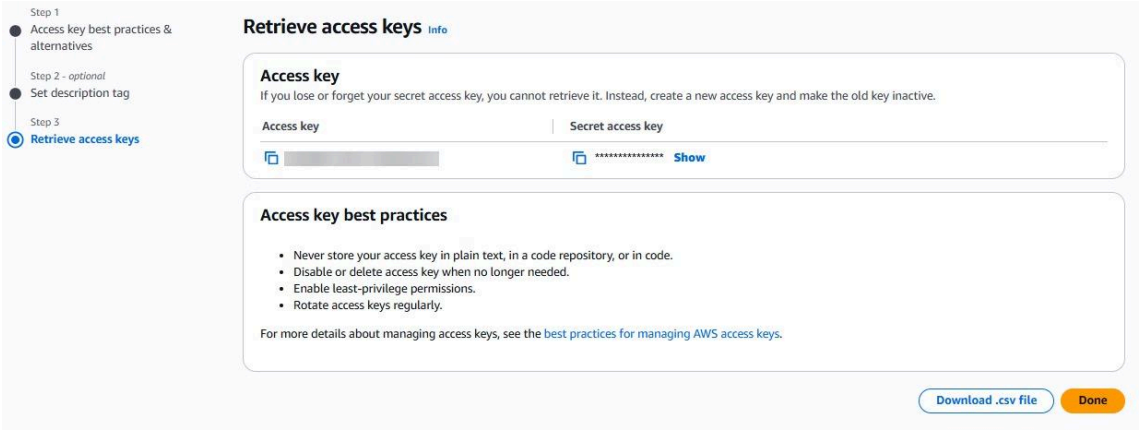
4. Setelah itu, selesaikan proses pembuatan pengguna dan konfirmasikan pembuatan pengguna tersebut.



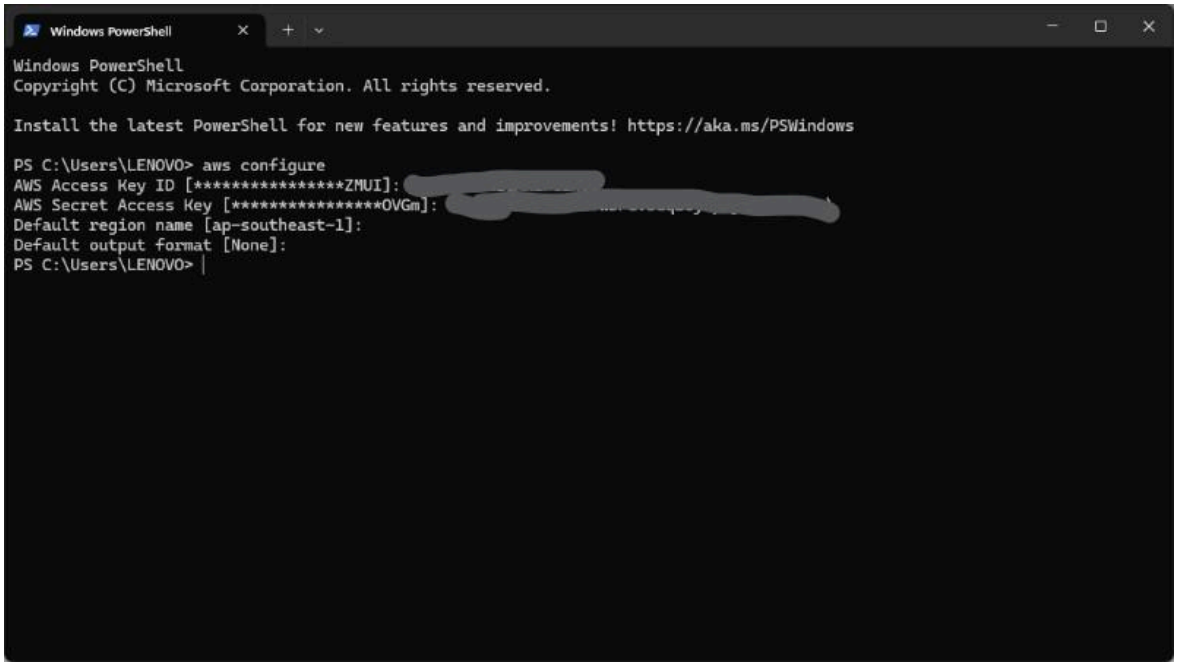


5. Buka halaman pengguna IAM yang baru dibuat, lalu klik Buat Kunci Akses. Pilih CLI untuk kasus penggunaan dan tambahkan tag deskripsi (opsional). Klik Buat, dan ambil Access Key ID dan Secret Access Key yang diberikan.



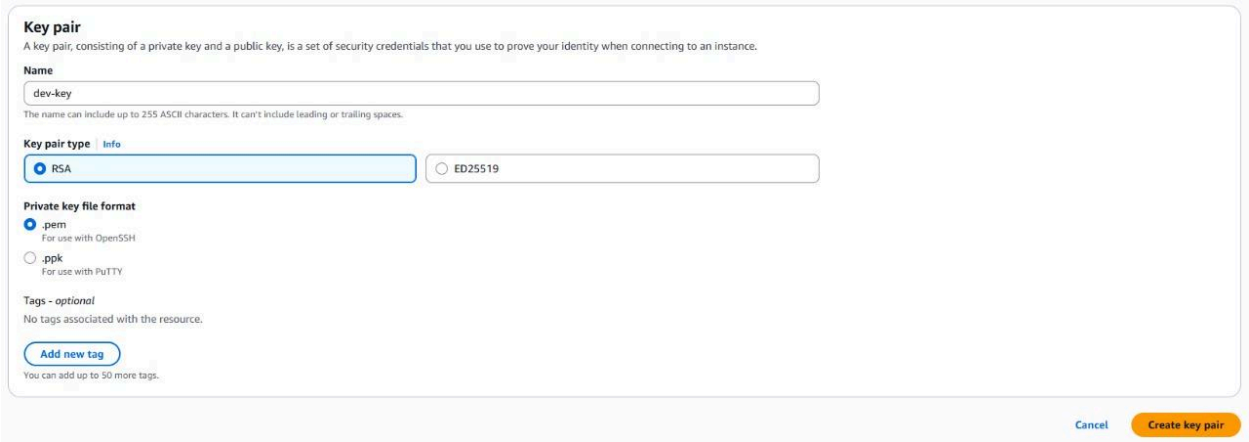


6. Terakhir, buka terminal dan jalankan perintah aws configure. Masukkan Access Key ID dan Secret Access Key saat diminta untuk menyelesaikan konfigurasi.



Set Up Key Pair for .tfvars (Used in Terraform):

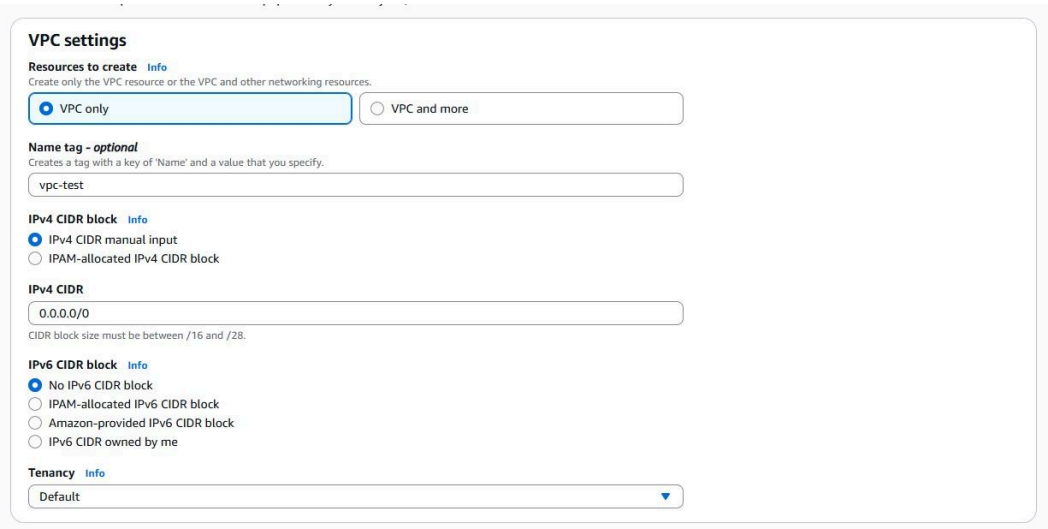
1. Masuk ke AWS Console dan buka layanan EC2.
2. Pada sidebar kiri, cari dan pilih Key Pairs.
3. Klik Buat Key Pair untuk memulai proses pembuatan.
4. Masukkan nama untuk key pair yang akan dibuat.
5. Pilih RSA sebagai jenis key pair.
6. Pilih .pem sebagai format file untuk key pair tersebut.
7. Klik Buat Key Pair untuk menyelesaikan proses.



Set Up VPC Services for .tfvars (Used in Terraform):

1. Masuk ke AWS Console dan buka layanan **VPC**.

2. Pada sidebar kiri, cari dan pilih **Your VPCs**.

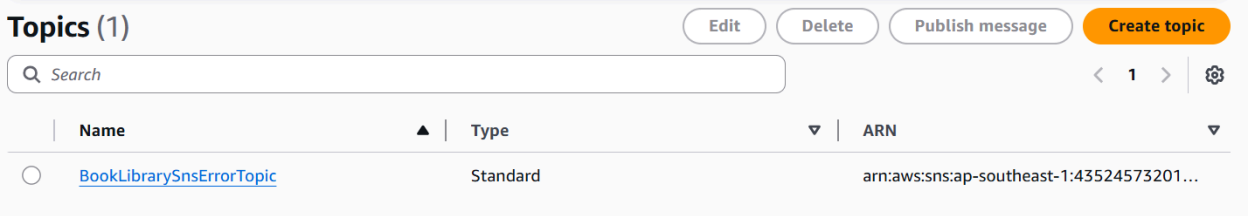


3. Klik **Buat VPC** untuk memulai pembuatan.
4. Pilih **VPC Only** untuk jenis sumber daya yang akan dibuat.
5. Berikan **Name Tag** untuk VPC yang akan dibuat.
6. Tentukan **IPv4 CIDR** (Anda bisa menggunakan default 10.0.0.0/24).
7. Klik **Buat** dan setelah itu salin **VPC ID** yang telah dibuat.

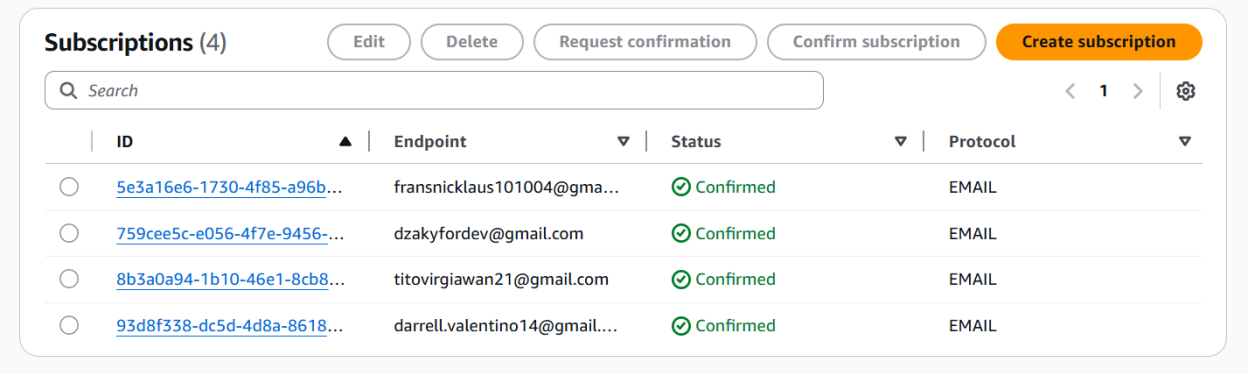
Set Up SNS for notification in case of error:

Untuk SNS ini lebih baik dibuat manual agar tidak harus memverifikasi subscription tiap email.

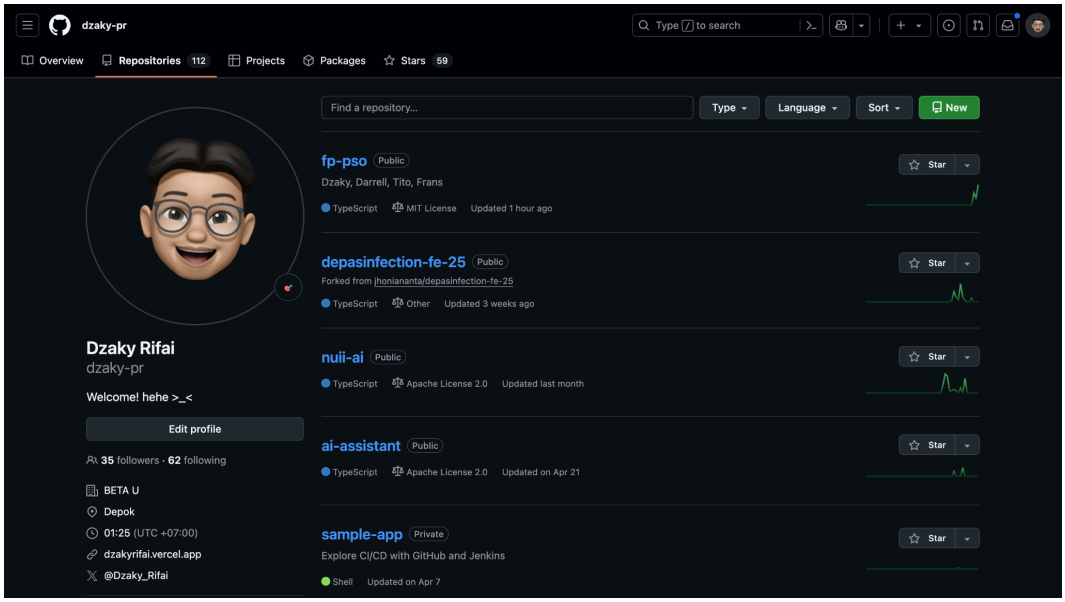
1. Masuk ke AWS Console dan buka layanan **SNS**.
2. Masuk ke tab **Topics** lalu klik tombol **Create Topic**
3. Untuk type nya pilih **Standard,** Lalu masukkan nama topic nya, misalkan "BookLibrarySnsErrorTopic" dan untuk Display Name nya dibebaskan saja atau opsional.
4. Lalu scroll ke bawah dan klik **Create Topic**
5. Balik ke tab topic dengan klik Topic di tab Topics, dan klik nama topic yang barusan dibuat.



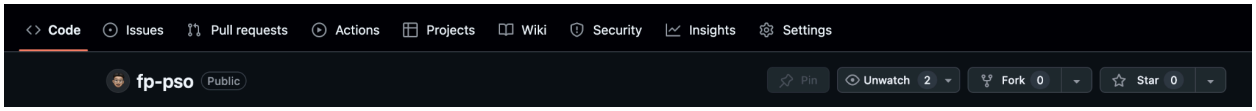
6. Scroll ke bawah dan klik tombol berwarna orange **Create Subscription**
7. Pada field protocol, pilih yang **"Email"**
8. Masukkan email yang mau mendapatkan notifikasi saat ada error
9. Klik **Create Subscription**
10. Buka email yang barusan dimasukkan dan buka mail dari AWS untuk verifikasi subscription, klik link tersebut
11. Balik ke dalam topic yang baru dibuat dan scroll ke bawah dan seharusnya email sudah menjadi status **Confirmed.**
12. Lakukan ke seberapa banyak email yang ingin terkoneksi.



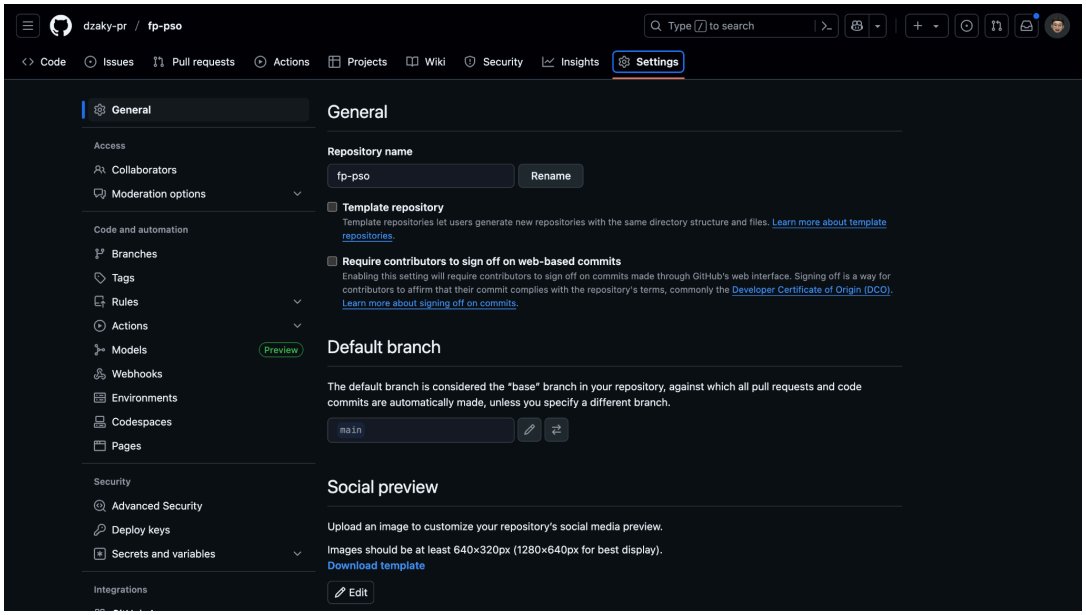
Set Up Github Secret :



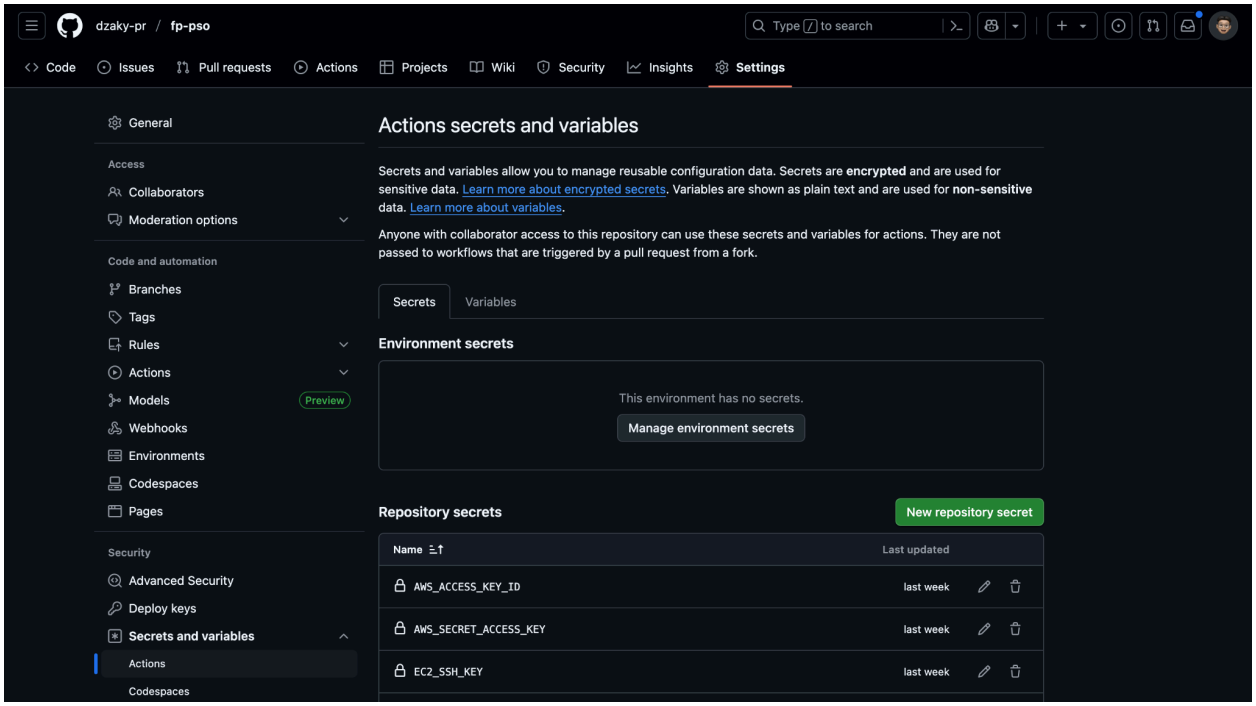
1. Masuk ke halaman repo yang akan pakai contohnya https://github.com/dzaky-pr/fp-pso



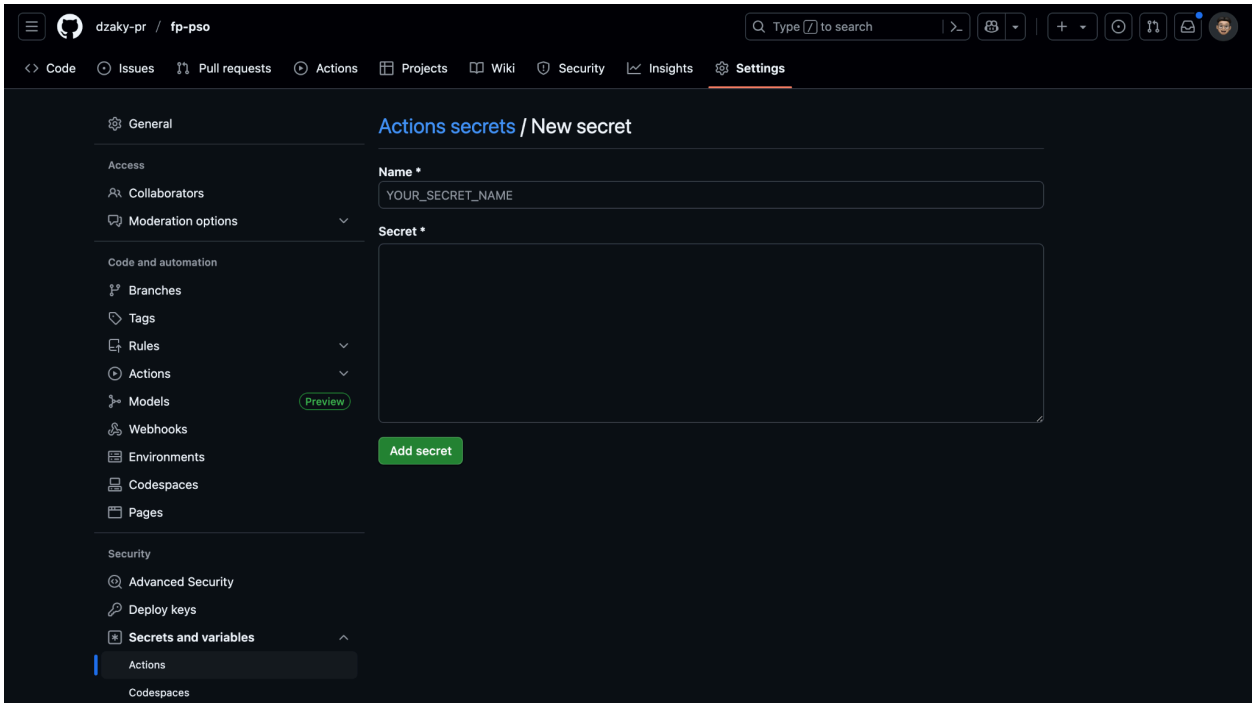
2. Pilih settings pada repository tersebut.



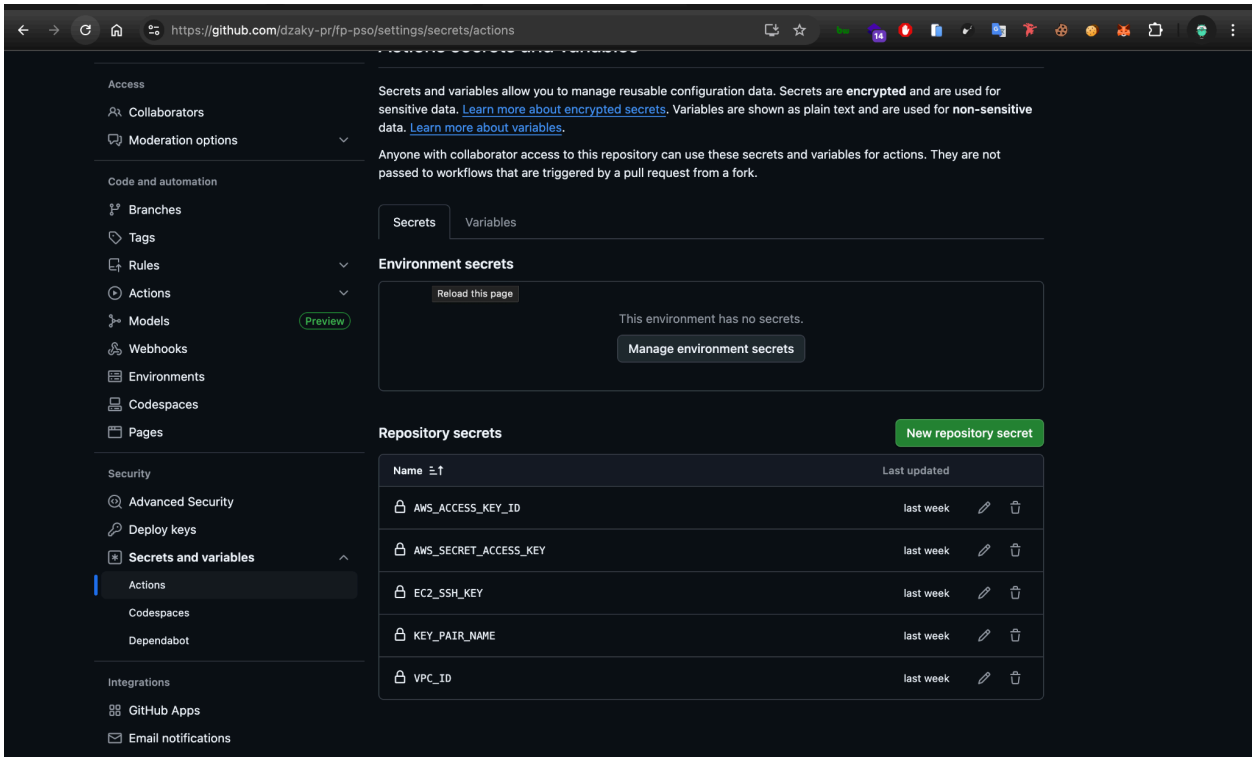
3. Pada navigasi dikiri, pilih “Secrets and variables” di bagian Security.



4. Pada contoh di atas sudah ada beberapa repository secrets. Untuk menambah secrets baru klik button hijau “New repository secret”

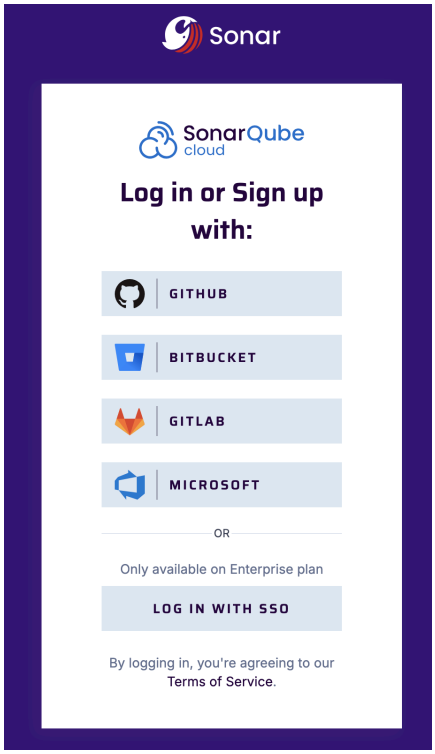


5. Masukkan Name dan Secretnya (pastikan sesuai dengan yang ada pada repository dan README petunjuk repo)

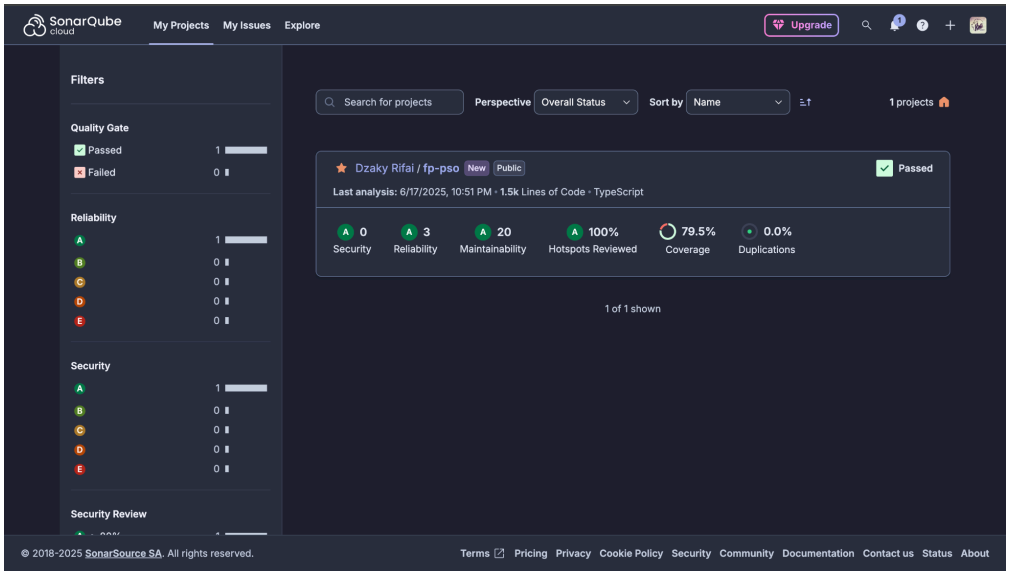


6. Jika berhasil, akan tampil Repository secrets baru.

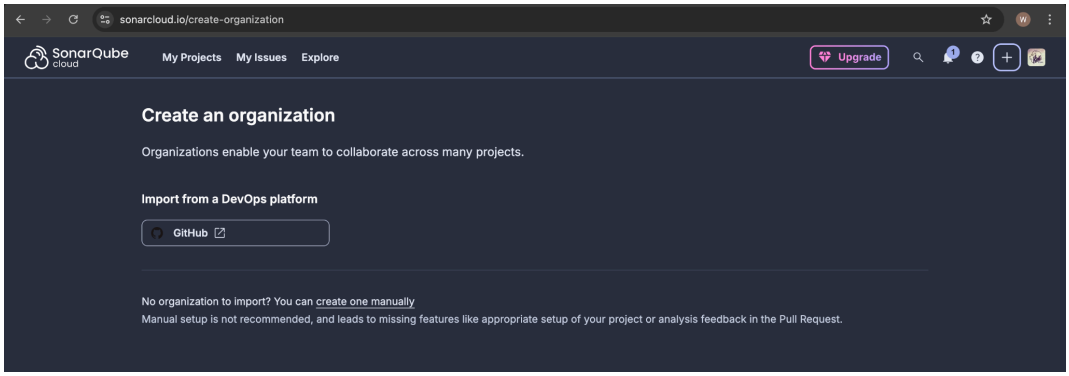
Penjelasan SonarQube Cloud



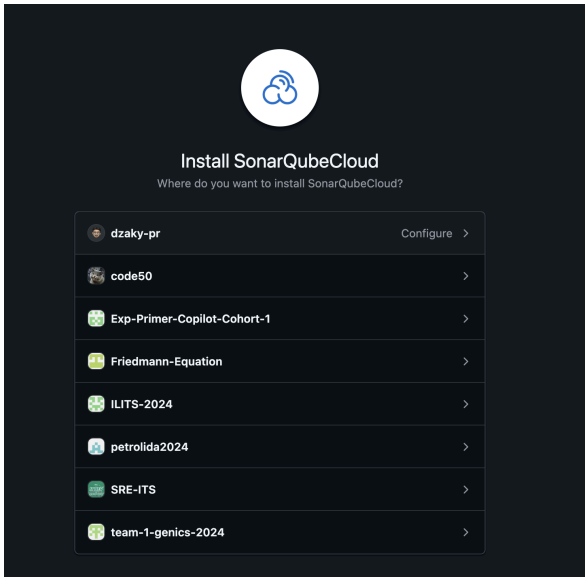
1. Masuk ke halaman https://sonarcloud.io/login dan login menggunakan Github



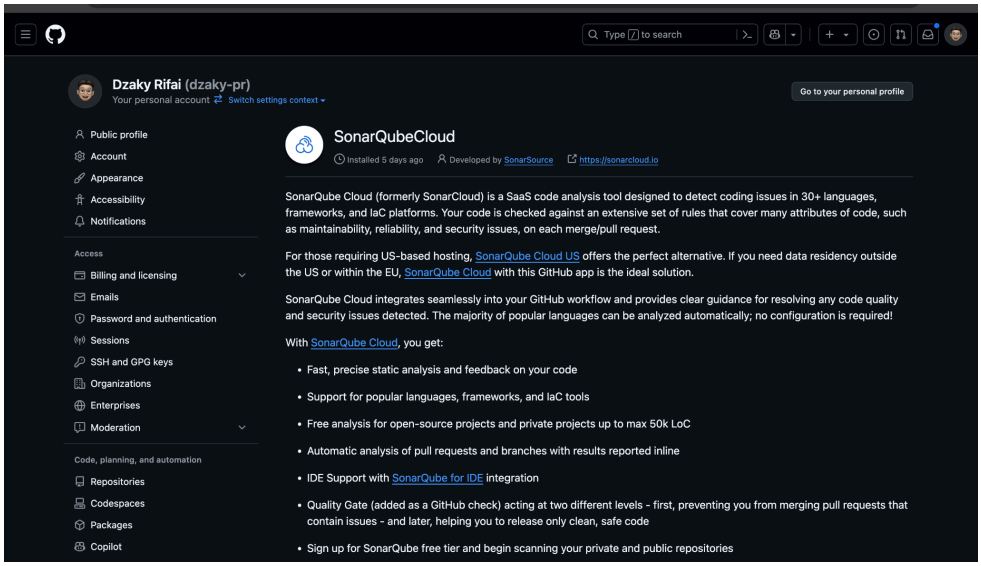
2. Di atas adalah contoh setelah setup seluruhnya. Namun karena ingin membuat baru, klik pojok kanan atas tombol “+” dan create organization (ini seperti di awal jika belum sama sekali punya sonarqube cloud)



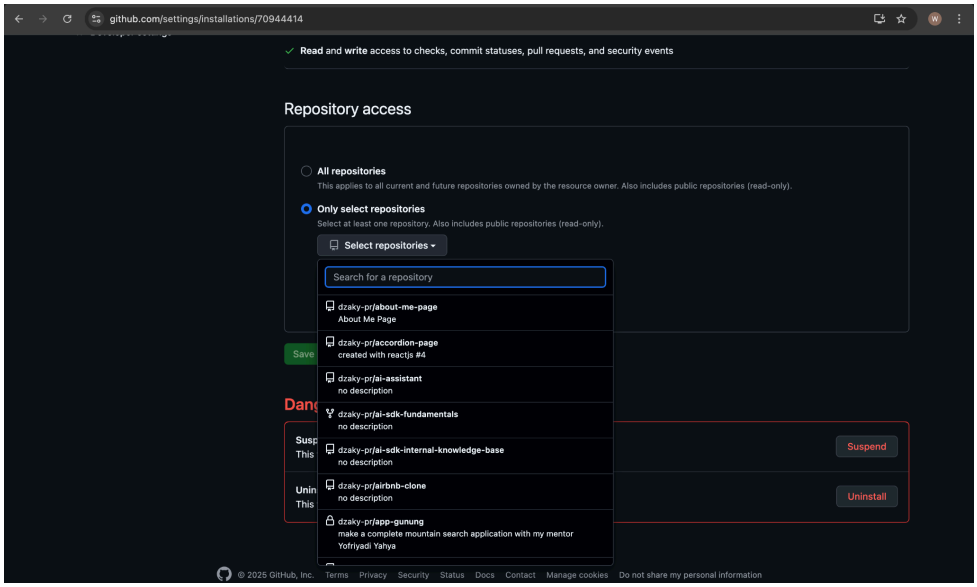
3. Pilih import from a DevOps platform Github



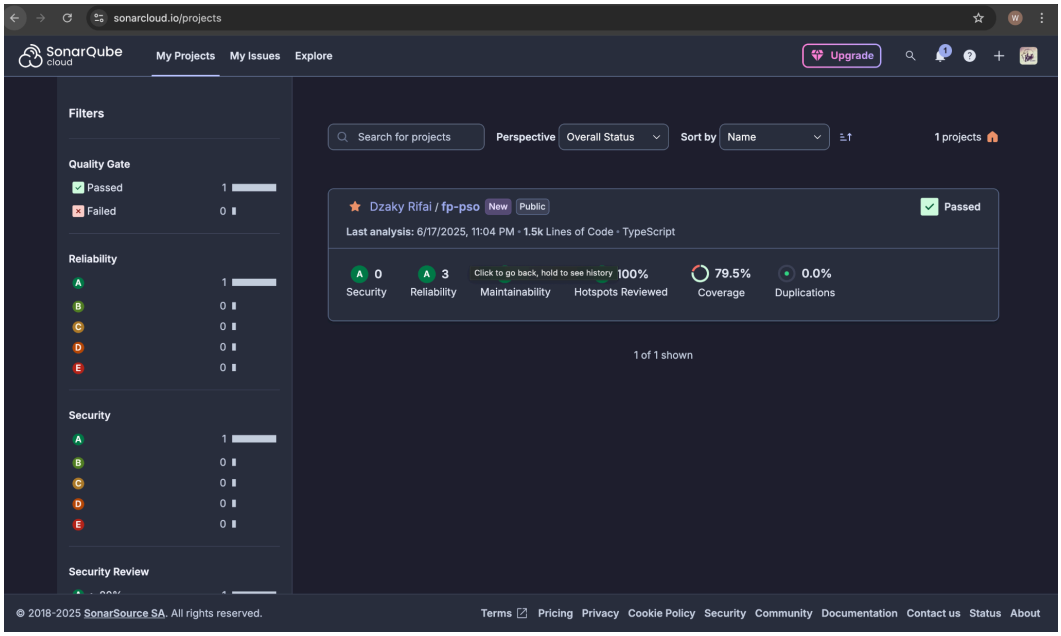
4. Contoh di atas saya sudah membuat organization pada dzaky-pr. Klik sesuai kebutuhan (contohnya disini pilih dzaky-pr)

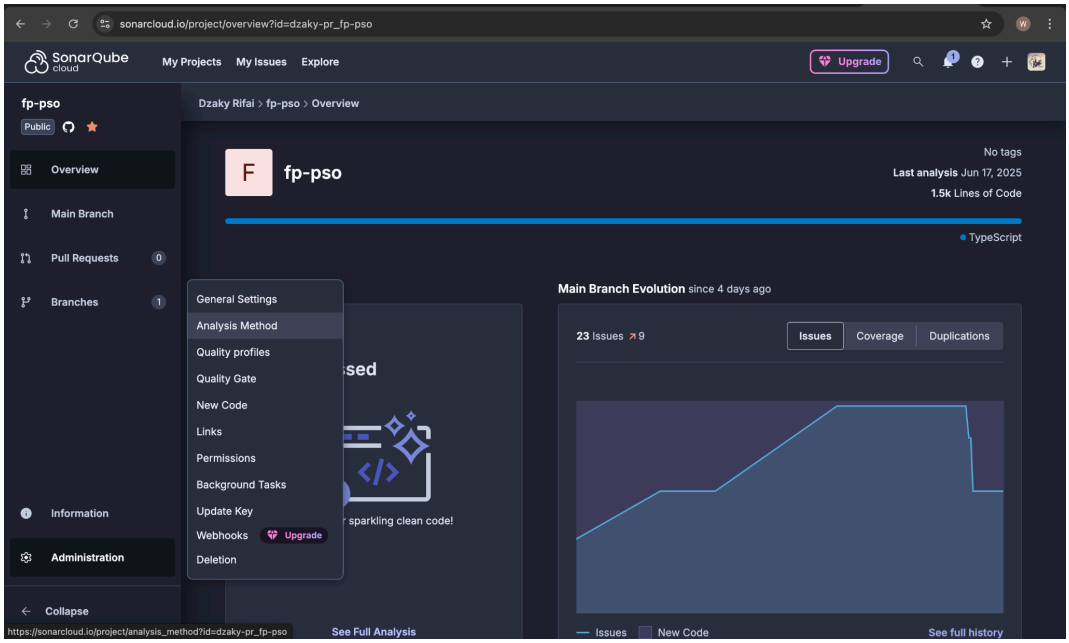


5. Jika sudah pada halaman di atas, scroll ke bawah.

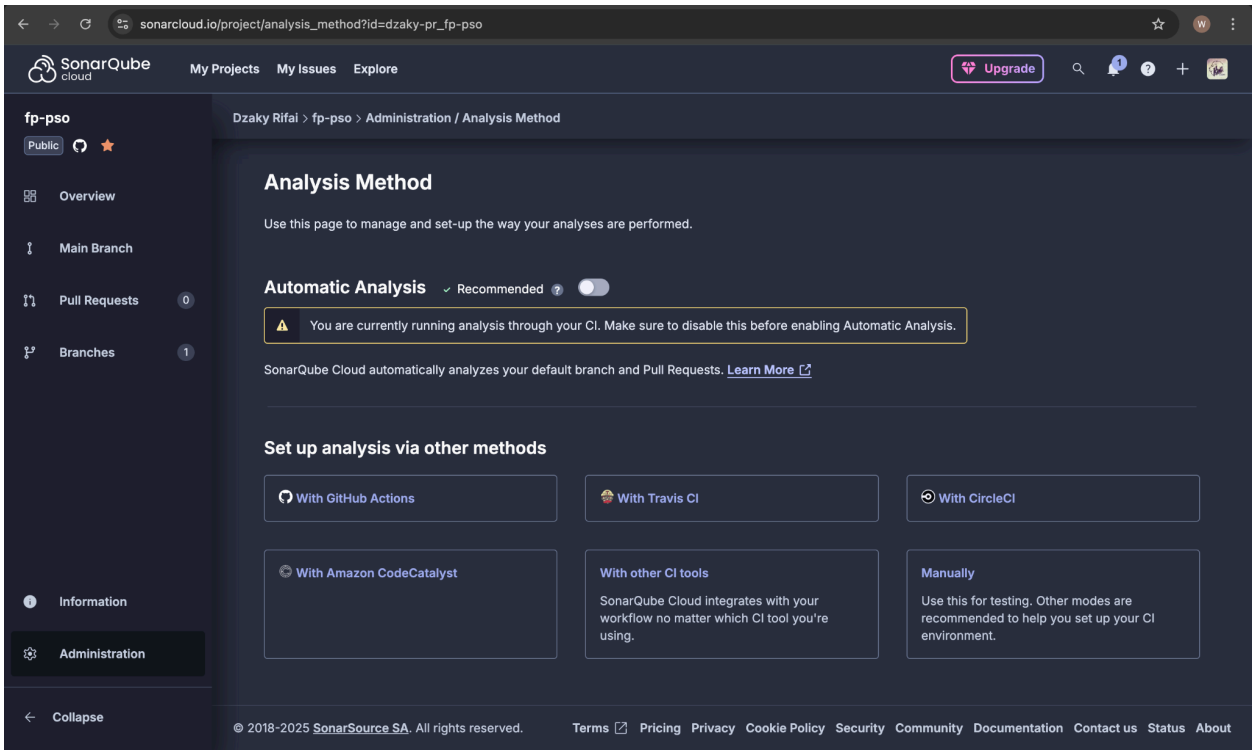


6. Pilih repository yang anda inginkan dan klik save. Setelah itu kembali ke https://sonarcloud.io/projects dan pilih repository dari organization yang sudah dipilih.





7. Pilih “Administration” di navigasi kiri dan pilih Analysis Method.



8. Pastikan Analysis Method off dan klik Set up analysis via other methods “With Github Actions”.

(selanjutnya saya tidak bisa menunjukkan karena akan memberikan value sensitif). Anda akan dipandu persis seperti https://docs.sonarsource.com/sonarqube-cloud/advanced-setup/ci-based-analysis/github-actions-for-sonarcloud/ yang saya sisipkan pada code CI bagian jest test dengan --coverage dan memasukkan github secrets SONARCLOUD_TOKEN.

Penjelasan Infrastruktur (Terraform)

Set Provider, Terraform Backend, and Suffix

```
provider "aws" {
  region = var.aws_region
}

terraform {
  backend "s3" {
    bucket         = "tf-state-bucket-booklibrary"
    key            = "terraform.tfstate"
    region         = "ap-southeast-1"
    encrypt        = true
  }
}

resource "random_id" "suffix" {
  # Random ID for unique resource names
  byte_length = 4
}
```

- provider "aws"
  Ini mengatur koneksi ke AWS, menentukan di region mana resource akan dibuat.
- terraform { backend "s3" { ... } }
  Ini mengkonfigurasi penyimpanan state Terraform di S3 bucket, penting untuk melacak resource yang sudah di deploy dan kolaborasi tim.
- resource "random_id" "suffix"
  Resource ini menghasilkan ID acak, biasanya digunakan sebagai suffix untuk membuat nama resource lain menjadi unik, mencegah konflik penamaan.

Set S3 Bucket untuk Artifact Storage

```
resource "aws_s3_bucket" "artifact" {
  bucket = "${var.artifact_bucket_name}-${random_id.suffix.hex}"
  force_destroy = true # Allows deletion of non-empty bucket

  tags = {
    Name        = "${var.artifact_bucket_name}-${random_id.suffix.hex}"
    Environment = "Dev"
    Project     = "BookLibrary"
  }
}

resource "aws_s3_bucket_versioning" "artifact" {
  bucket = aws_s3_bucket.artifact.id
  versioning_configuration {
    status = "Enabled"
  }
}
```

- resource "aws_s3_bucket" "artifact"
  Resource ini membuat S3 bucket baru di AWS. Nama bucket (bucket) dan tag Name dibentuk dari variabel artifact_bucket_name ditambah ID acak (random_id.suffix.hex), memastikan nama bucket unik. force_destroy = true memungkinkan penghapusan bucket meskipun tidak kosong. Tags seperti Environment dan Project digunakan untuk pengorganisasian dan identifikasi resource.
- resource "aws_s3_bucket_versioning" "artifact"
  Resource ini mengaktifkan versioning untuk S3 bucket yang dibuat sebelumnya (aws_s3_bucket.artifact). Dengan versioning diaktifkan (status = "Enabled"), setiap kali objek dalam bucket dimodifikasi atau dihapus, versi baru objek akan disimpan, bukan menimpa yang lama. Ini berguna untuk pemulihan dari penghapusan tidak sengaja atau perubahan data.

Set S3 Bucket untuk Lambda Function

```
resource "aws_s3_bucket" "lambda_bucket" {
  bucket = "${var.api_bucket_name}-${random_id.suffix.hex}"

  tags = {
    Name        = "${var.api_bucket_name}-${random_id.suffix.hex}"
    Environment = "Dev"
    Project     = "BookLibrary"
  }
}
```

Resource ini membuat S3 bucket baru di AWS, khusus untuk menyimpan artefak atau kode yang terkait dengan fungsi Lambda. Nama bucket (bucket) dan tag Name dibuat unik dengan menggabungkan variabel api_bucket_name dan ID acak (random_id.suffix.hex). Tags seperti Environment dan Project membantu dalam kategorisasi dan pengelolaan bucket ini dalam proyek.

Set Resource to Build Lambda Function

```
resource "null_resource" "build_lambda" {
  triggers = {
    lambda_source = filemd5("${path.module}/lambda.js")
    package_json = filemd5("${path.module}/../package.json")
  }

  provisioner "local-exec" {
    command = "cd ${path.module}/.. && npm run build:lambda"
  }
}

data "archive_file" "lambda_zip" {
  depends_on = [null_resource.build_lambda]

  type        = "zip"
  source_file = "${path.module}/dist/lambda.js"
  output_path = "${path.module}/build/lambda_package.zip"
}

resource "aws_s3_object" "lambda_code_upload" {
  bucket = aws_s3_bucket.lambda_bucket.id
  key    = "lambda_package.zip"
  source = data.archive_file.lambda_zip.output_path
  etag   = filemd5(data.archive_file.lambda_zip.output_path)
}
```

- resource "null_resource" "build_lambda"
  Ini adalah resource "placeholder" yang tidak membuat objek di cloud. Tujuannya adalah untuk menjalankan local-exec provisioner ketika ada perubahan pada file-file yang dilacak oleh triggers (misalnya, lambda.js atau package.json). Dalam kasus ini, provisioner menjalankan perintah npm run

build:lambda untuk mengkompilasi atau menyiapkan kode Lambda secara lokal.

- data "archive_file" "lambda_zip"
  Data block ini digunakan untuk membuat file arsip (ZIP). Setelah build_lambda selesai (karena depends_on), data ini mengambil kode Lambda yang telah dikompilasi (source_file) dan mengarsipkannya ke lokasi output_path (lambda_package.zip).
- resource "aws_s3_object" "lambda_code_upload"
  Resource ini mengunggah file ZIP hasil arsip (data.archive_file.lambda_zip.output_path) ke S3 bucket yang ditentukan (aws_s3_bucket.lambda_bucket.id). Ini adalah langkah terakhir untuk menempatkan kode Lambda di lokasi yang dapat diakses oleh AWS Lambda. etag dihitung untuk memastikan integritas file yang diunggah.

Set DynamoDB for Books Table

```
resource "aws_dynamodb_table" "books_table" {
  name         = "${var.table_name}-${random_id.suffix.hex}"
  billing_mode = "PAY_PER_REQUEST"
  hash_key     = "id"

  attribute {
    name = "id"
    type = "N" # N for Number, based on lambda.js [cite: 32, 34]
  }

  tags = {
    Name        = "${var.table_name}-${random_id.suffix.hex}"
    Environment = "Dev"
    Project     = "BookLibrary"
  }
}
```

Resource ini membuat DynamoDB table baru bernama books_table. Nama tabel (name) dibuat unik dengan kombinasi var.table_name dan ID acak. billing_mode

diatur ke PAY_PER_REQUEST, artinya Anda membayar sesuai penggunaan. hash_key adalah id dengan tipe data Number (N), berfungsi sebagai primary key untuk tabel ini. Tags digunakan untuk identifikasi dan pengelolaan.

Set DynamoDB for Users Table

```
resource "aws_dynamodb_table" "users_table" {
  name         = "users-${random_id.suffix.hex}"
  billing_mode = "PAY_PER_REQUEST"
  hash_key     = "userId"

  attribute {
    name = "userId"
    type = "S"
  }

  attribute {
    name = "email"
    type = "S" # Tipe data string untuk email
  }

  global_secondary_index {
    name               = "EmailIndex"
    hash_key           = "email"
    projection_type    = "ALL"

    read_capacity      = 1
    write_capacity     = 1
  }

  tags = {
    Name        = "users-table-book-library-${random_id.suffix.hex}"
    Environment = "Dev" # Sesuaikan dengan tag lingkungan Anda
    Project     = "BookLibrary"
  }
}
```

Resource ini membuat DynamoDB table lain bernama users_table. Nama tabelnya juga unik. Ini juga menggunakan billing_mode PAY_PER_REQUEST. hash_key adalah userId dengan tipe String (S). Selain itu, tabel ini memiliki global_secondary_index bernama EmailIndex pada atribut email (tipe String, S). Indeks ini memungkinkan pencarian efisien berdasarkan email. Tags juga diterapkan untuk pengorganisasian.

Set IAM User for CI Purpose in GitHub Actions

```
resource "aws_iam_user" "ci_user" {
  name = "github-actions-cli-ci-${random_id.suffix.hex}"
}

resource "aws_iam_user_policy" "ci_policy" {
  name   = "ci-access-policy-${random_id.suffix.hex}"
  user   = aws_iam_user.ci_user.name
  policy = data.aws_iam_policy_document.ci_doc.json
}

data "aws_iam_policy_document" "ci_doc" {
  statement {
    actions = [
      "s3:PutObject",
      "s3:GetObject",
      "s3:PutObjectAcl",
      "s3:ListBucket",
      "s3:DeleteObject"
    ]
    resources = [
      aws_s3_bucket.artifact.arn,
      "${aws_s3_bucket.artifact.arn}/*"
    ]
  }
}
```

- resource "aws_iam_user" "ci_user"
  Resource ini membuat IAM user baru di AWS, yang dinamai secara unik dengan menggabungkan "github-actions-cli-ci-" dan ID acak. User ini kemungkinan akan digunakan oleh alur kerja CI/CD (Continuous Integration/Continuous Deployment), seperti GitHub Actions, untuk berinteraksi dengan layanan AWS.

- data "aws_iam_policy_document" "ci_doc"
  Data block ini membuat dokumen IAM policy secara dinamis. Policy ini mendefinisikan izin yang spesifik:
  - **actions**: Memberikan izin untuk melakukan operasi S3 seperti mengunggah (`PutObject`), mengambil (`GetObject`), mengatur ACL (`PutObjectAcl`), mencantumkan `bucket` (`ListBucket`), dan menghapus objek (`DeleteObject`).
  - **resources**: Membatasi izin ini hanya untuk `S3 bucket artifact` tertentu (`aws_s3_bucket.artifact.arn`) dan semua objek di dalamnya (`${aws_s3_bucket.artifact.arn}/*`).
- resource "aws_iam_user_policy" "ci_policy"
  Resource ini melampirkan IAM policy yang telah didefinisikan sebelumnya (data.aws_iam_policy_document.ci_doc) ke IAM user yang baru dibuat (aws_iam_user.ci_user). Nama policy ini juga unik. Dengan demikian, ci_user sekarang memiliki izin yang diperlukan untuk berinteraksi dengan S3 bucket artifact sesuai dengan operasi yang diizinkan.

Set IAM User for CD Purpose in GitHub Actions

```
resource "aws_iam_user" "cd_user" {
  name = "github-actions-cli-cd-${random_id.suffix.hex}"
}

resource "aws_iam_user_policy" "cd_policy" {
  name   = "cd-access-policy-${random_id.suffix.hex}"
  user   = aws_iam_user.cd_user.name
  policy = data.aws_iam_policy_document.cd_doc.json
}
```

- resource "aws_iam_user" "cd_user"
  Resource ini membuat IAM user baru di AWS. Nama user ini unik karena digabungkan dengan ID acak (random_id.suffix.hex). User ini secara spesifik

ditujukan untuk digunakan oleh sistem CD (Continuous Deployment) seperti GitHub Actions untuk melakukan operasi deployment pada resource AWS.

- resource "aws_iam_user_policy" "cd_policy"
  Resource ini berfungsi untuk melampirkan sebuah policy izin ke IAM user cd_user yang baru dibuat. Nama policy ini juga unik. Policy izin yang dilampirkan (data.aws_iam_policy_document.cd_doc.json) akan mendefinisikan secara spesifik action apa saja yang diizinkan untuk dilakukan oleh cd_user terhadap resource AWS yang relevan dengan proses deployment.

Set IAM User Policy for CD User

```
data "aws_iam_policy_document" "cd_doc" {
  statement {
    sid     = "ReadOnlyS3ArtifactBucket"
    effect  = "Allow"

    actions = [
      "s3:PutObject",
      "s3:GetObject",
      "s3:PutObjectAcl",
      "s3:ListBucket"
    ]

    resources = [
      aws_s3_bucket.artifact.arn,
      "${aws_s3_bucket.artifact.arn}/*"
    ]
  }

  statement {
    sid     = "OptionalEC2AndELBAccess"
    effect  = "Allow"

    actions = [
      "ec2:DescribeInstances",
      "ec2:DescribeTags",
      "elasticloadbalancing:Describe*",
      "elasticloadbalancing:ModifyListener",
      "elasticloadbalancing:ModifyRule"
    ]

    resources = ["*"]
  }
}
```

Data block ini membuat dua statement dalam satu IAM policy document untuk CD (Continuous Deployment).

- **statement pertama (SID: `ReadOnlyS3ArtifactBucket`)**:
  - Memberikan izin `Allow` untuk operasi S3 tertentu (`PutObject`, `GetObject`, `PutObjectAcl`, `ListBucket`).
  - Izin ini dibatasi hanya pada `S3 bucket` yang digunakan untuk `artifact` (`aws_s3_bucket.artifact.arn`) dan semua objek di dalamnya. Ini

memungkinkan sistem `CD` untuk membaca dan menulis ke `bucket` `artifact`.

- **statement kedua (SID: `OptionalEC2AndELBAccess`)**:
  - Memberikan izin `Allow` untuk operasi terkait EC2 dan Elastic Load Balancing (ELB).
  - Termasuk izin untuk `DescribeInstances` (melihat instance EC2), `DescribeTags` (melihat `tag` EC2), serta berbagai `action` `Modify` pada `ELB listener` dan `rule`.
  - `resources = ["*"]` berarti izin ini berlaku untuk semua `resource` EC2 dan ELB, yang mungkin perlu dipersempit tergantung kebutuhan spesifik `deployment`. Ini mengindikasikan bahwa `CD` memerlukan akses untuk mengelola atau melihat status `instance` dan konfigurasi `load balancer`.

Set IAM Role to be used with Lambda

```
resource "aws_iam_role" "lambda_exec_role" {
  name = "${var.lambda_function_name}-role-${random_id.suffix.hex}"

  assume_role_policy = jsonencode({
    Version   = "2012-10-17"
    Statement = [{
      Action    = "sts:AssumeRole"
      Effect    = "Allow"
      Principal = {
        Service = "lambda.amazonaws.com"
      }
    }]
  })
}
```

Resource ini membuat IAM role di AWS yang dirancang khusus untuk dieksekusi oleh fungsi Lambda.

- name: Nama `role` ini unik, dibentuk dari nama fungsi Lambda dan ID acak.
- assume_role_policy: Bagian ini mengatur siapa atau layanan apa yang diizinkan untuk menggunakan `role` ini. Dalam kasus ini, `assume_role_policy`

mengizinkan layanan AWS Lambda (`lambda.amazonaws.com`) untuk mengambil peran ini. Jadi, fungsi Lambda bisa menggunakan izin yang melekat pada `role` ini untuk berinteraksi dengan layanan AWS lainnya.

Penjelasan CI Pipeline

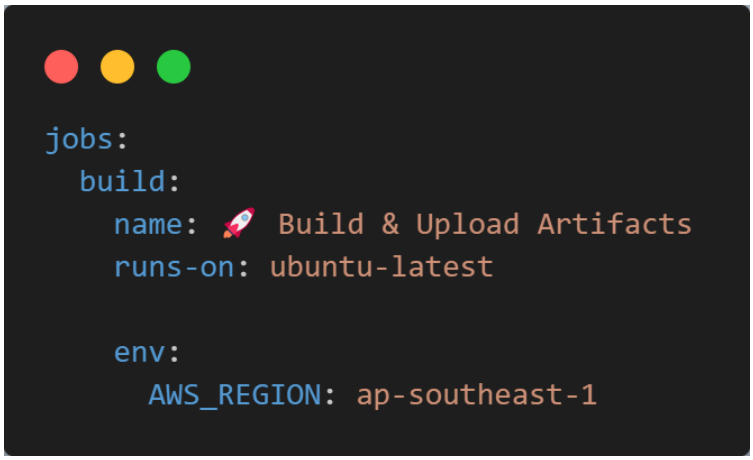
Inisialisasi pipeline

```yaml
name: "🛠 CI Pipeline"

on:
  workflow_dispatch:
  push:
    branches:
      - main

concurrency:
  group: ci-${{ github.ref }}
  cancel-in-progress: true
```

Awal-awal pipeline menuliskan nama pipeline kita, dalam kasus ini adalah CI Pipeline. Lalu syntax on: menyatakan pipeline akan jalan saat kapan :

1. Workflow_dispatch : berarti workflow bisa dijalankan secara manual menggunakan GitHub Actions
2. Push: Branches : -main berarti pipeline akan berjalan setiap kali ada push pada branch main
3. Concurrency : berarti pipeline akan berhenti jika sedang berjalan jika sebuah pipeline akan dijalankan bersamaan agar tidak konflik.

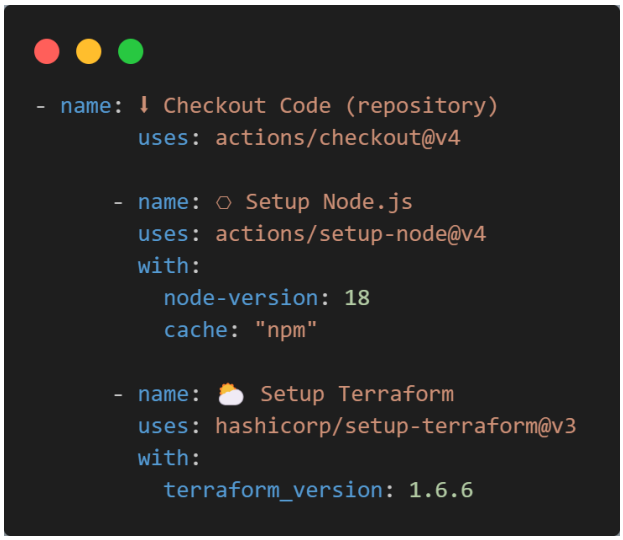
Configurasi Job



```yaml
jobs:
  build:
    name: 🚀 Build & Upload Artifacts
    runs-on: ubuntu-latest

    env:
      AWS_REGION: ap-southeast-1
```

Kode ini sebagai dasar untuk jobs yang akan dilakukan. Selanjutnya memastikan semua job dijalankan menggunakan ubuntu-latest dengan syntax runs-on. Lalu mengset environment variables, yaitu ap-southeast-1 untuk setiap jobs yang dijalankan.

Set Up Node and Terraform



```yaml
- name: ⬇ Checkout Code (repository)
        uses: actions/checkout@v4

      - name: ⬡ Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: 18
          cache: "npm"

      - name: 🌥 Setup Terraform
        uses: hashicorp/setup-terraform@v3
        with:
          terraform_version: 1.6.6
```

Langkah pertama dari pipeline adalah checkout code dari repo kami. Selanjutnya adalah Menginstall Node.js versi 18 di runner menggunakan actions/setup-node@v4, serta mengaktifkan cache npm untuk mempercepat instalasi dependensi. Lalu menginstall Terraform versi 1.6.6 di runner menggunakan hashicorp/setup-terraform@v3.

Terraform Init serta cek output

```yaml
- name: 🌱 Terraform Init
        run: terraform -chdir=./terraform init
        env:
          AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
          AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          AWS_REGION: ${{ env.AWS_REGION }}

      - name: 📦 Terraform Output
        id: tf_output
        run: |
          terraform -chdir=./terraform output -json > tf_output.json

          # Mask sensitive values in logs
          CI_ACCESS_KEY=$(jq -r '.ci_access_key_id.value' tf_output.json)
          CI_SECRET_KEY=$(jq -r '.ci_secret_access_key.value' tf_output.json)
          echo "::add-mask::$CI_ACCESS_KEY"
          echo "::add-mask::$CI_SECRET_KEY"

          echo "api_gateway_url=$(jq -r '.api_gateway_url.value' tf_output.json)" >> $GITHUB_OUTPUT
          echo "artifact_bucket=$(jq -r '.artifact_bucket_name.value' tf_output.json)" >> $GITHUB_OUTPUT
          echo "ci_access_key_id=$CI_ACCESS_KEY" >> $GITHUB_OUTPUT
          echo "ci_secret_access_key=$CI_SECRET_KEY" >> $GITHUB_OUTPUT
        env:
          AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
          AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          AWS_REGION: ${{ env.AWS_REGION }}

      - name: ✅ Check Terraform Output
        run: |
          echo "API Gateway URL: ${{ steps.tf_output.outputs.api_gateway_url }}"
          echo "Artifact Bucket: ${{ steps.tf_output.outputs.artifact_bucket }}"
```

Bagian workflow ini menjalankan inisialisasi Terraform dengan perintah terraform init di direktori terraform, menggunakan kredensial AWS yang diambil dari secrets GitHub. Setelah itu, workflow mengambil output dari Terraform dalam format JSON dan menyimpannya ke file tf_output.json. Nilai-nilai sensitif seperti access key dan secret key diambil dari output tersebut, kemudian di-mask agar tidak muncul di log publik. Output penting seperti URL API Gateway dan nama bucket artifact juga diekstrak dan disimpan ke environment output GitHub Actions agar bisa digunakan di langkah berikutnya. Terakhir, workflow menampilkan nilai output utama (API Gateway URL dan nama bucket) ke log untuk memudahkan verifikasi hasil provisioning.

Terraform Init serta cek output

```yaml
- name: 📦 Install Dependencies
        run: |
          # Install with audit and cache optimization
          npm ci --prefer-offline --no-audit --silent

      - name: 🔍 Lint & Typecheck
        run: |
          npm run lint
          npm run typecheck

      - name: 🧪 Run Tests & Generate Coverage
        run: npm test -- --coverage --verbose

      - name: 📄 Show coverage files (Debug purpose)
        run: ls -lh coverage && head -n 20 coverage/lcov.info

      - name: 🔍 SonarCloud Scan
        uses: SonarSource/sonarcloud-github-action@v2
        with:
          args: >
            -Dsonar.projectKey=dzaky-pr_fp-pso
            -Dsonar.organization=dzaky-pr
            -Dsonar.javascript.lcov.reportPaths=coverage/lcov.info
        env:
          SONAR_TOKEN: ${{ secrets.SONARCLOUD_TOKEN }}
```

Bagian workflow ini bertugas untuk menyiapkan dan memeriksa kualitas kode aplikasi. Pertama, dependensi proyek diinstal menggunakan npm ci dengan opsi optimalisasi cache dan tanpa audit untuk mempercepat proses. Setelah itu, workflow menjalankan linting dan type checking untuk memastikan kode bebas dari error sintaks dan masalah tipe data. Selanjutnya, seluruh test dijalankan dengan perintah npm test dan hasil cakupan (coverage) kode dihasilkan. Untuk keperluan debugging, isi folder dan file coverage ditampilkan sebagian ke log. Terakhir, workflow melakukan analisis kualitas kode menggunakan SonarCloud, dengan mengirimkan hasil coverage ke layanan tersebut agar dapat dipantau dan dianalisis secara otomatis. Token autentikasi SonarCloud diambil dari secrets repository.

Setup and build App

```
- name: 📥 Setup .env
        run: |
          echo "AWS_API_URL=${{ steps.tf_output.outputs.api_gateway_url }}" >> .env

      - name: 📥 Build Next.js App
        run: npm run build

      - name: 📦 Prepare Package Artifacts
        run: |
          mkdir -p artifact
          cp -r .next/static .next/standalone/.next
          cp -r .next/standalone artifact/
```

Bagian workflow ini bertujuan untuk menyiapkan environment dan membangun aplikasi sebelum proses deployment atau upload artifact. Pertama, workflow membuat file .env dan mengisi variabel AWS_API_URL dengan nilai URL API Gateway yang dihasilkan dari output Terraform, sehingga aplikasi bisa terhubung ke backend yang benar. Selanjutnya, aplikasi Next.js dibuild menggunakan perintah npm run build untuk menghasilkan file produksi. Setelah build selesai, workflow menyiapkan folder artifact dan menyalin hasil build (.next/standalone dan static files) ke dalam folder tersebut, sehingga siap untuk proses deployment atau upload ke storage/artifact repository.

Zip Artifacts and upload to bucket

```yaml
- name: 📦 Zip Artifacts
        run: |
          zip -r artifact.zip artifact

      - name: 🔍 Verify Artifact Integrity
        run: |
          # Generate checksum for artifact
          sha256sum artifact.zip > artifact.zip.sha256
          echo "Artifact size: $(ls -lh artifact.zip | awk '{print $5}')"

          # Verify minimum size (should be > 1MB for a real deployment)
          ARTIFACT_SIZE=$(stat -f%z artifact.zip 2>/dev/null || stat -c%s artifact.zip)
          if [ "$ARTIFACT_SIZE" -lt 1048576 ]; then
            echo "::warning::Artifact size seems small ($ARTIFACT_SIZE bytes)"
          fi

      - name: ☁️ Upload Artifact Zip to S3
        run: |
          aws s3 cp artifact.zip s3://${{ steps.tf_output.outputs.artifact_bucket }}/${{ github.sha }}.zip
        env:
          AWS_ACCESS_KEY_ID: ${{ steps.tf_output.outputs.ci_access_key_id }}
          AWS_SECRET_ACCESS_KEY: ${{ steps.tf_output.outputs.ci_secret_access_key }}
          AWS_REGION: ${{ env.AWS_REGION }}
```

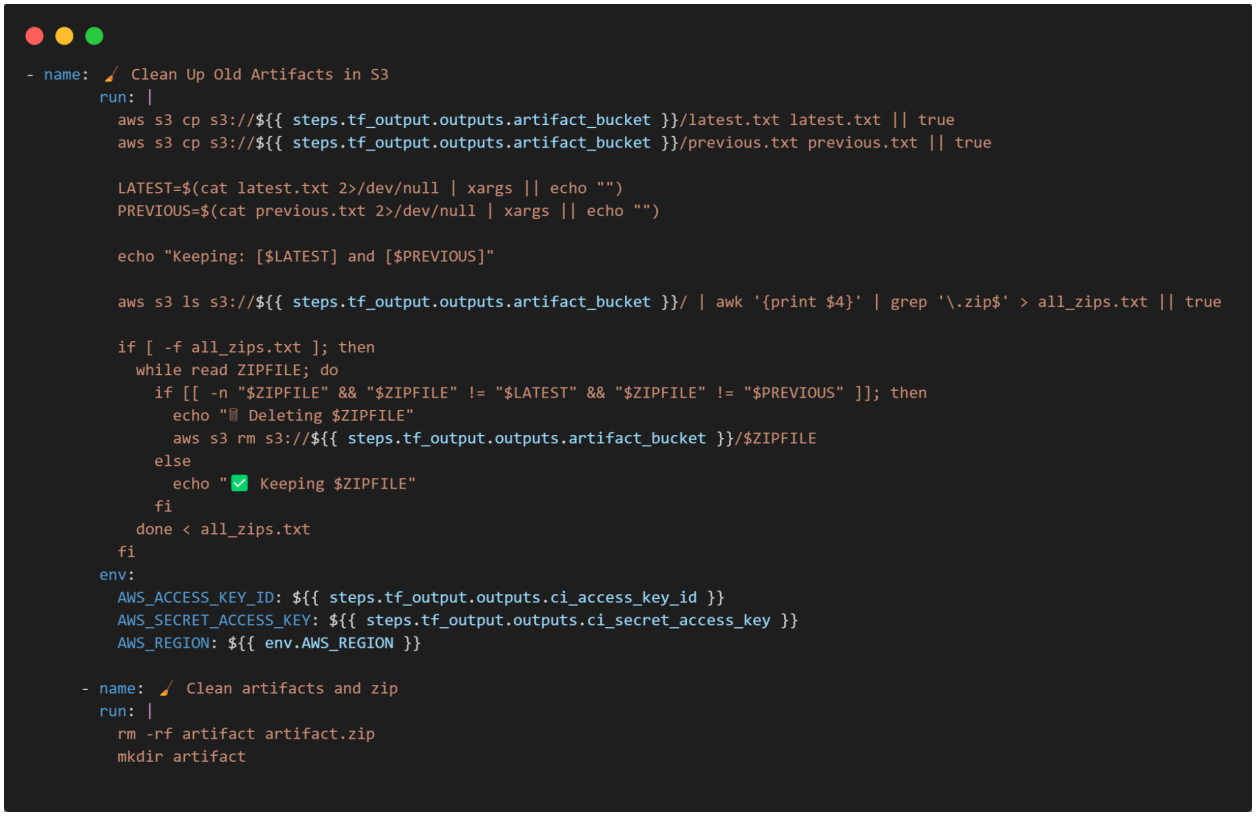
Bagian workflow ini bertugas untuk mengarsipkan hasil build aplikasi dan memastikan file artifact siap untuk di-upload ke storage. Pertama, seluruh isi folder artifact dikompres menjadi file artifact.zip. Setelah itu, workflow memverifikasi integritas artifact dengan membuat file checksum SHA256 dan menampilkan ukuran file zip ke log. Ada juga pengecekan otomatis untuk memastikan ukuran artifact tidak terlalu kecil (kurang dari 1MB), yang bisa menjadi indikasi build gagal atau file tidak lengkap. Jika ukuran terlalu kecil, workflow akan memberikan warning di log. Terakhir, file zip hasil build di-upload ke bucket S3 yang sudah ditentukan, menggunakan kredensial dan nama bucket yang diambil dari output Terraform, sehingga artifact siap digunakan untuk proses deployment selanjutnya.

Backup version

```yaml
- name: 📄 Backup latest.txt to previous.txt (if it exists)
        run: |
          set +e
          aws s3 cp s3://${{ steps.tf_output.outputs.artifact_bucket }}/latest.txt ./latest.txt
          if [ -f latest.txt ]; then
            echo "✅ Found latest.txt, backing up to previous.txt"
            aws s3 cp ./latest.txt s3://${{ steps.tf_output.outputs.artifact_bucket }}/previous.txt
          else
            echo "ℹ️ No latest.txt found, skipping backup."
          fi
        env:
          AWS_ACCESS_KEY_ID: ${{ steps.tf_output.outputs.ci_access_key_id }}
          AWS_SECRET_ACCESS_KEY: ${{ steps.tf_output.outputs.ci_secret_access_key }}
          AWS_REGION: ${{ env.AWS_REGION }}

      - name: 📄 Write and Upload latest.txt
        run: |
          echo "${{ github.sha }}.zip" > latest.txt
          aws s3 cp latest.txt s3://${{ steps.tf_output.outputs.artifact_bucket }}/latest.txt
          echo "✅ latest.txt now points to ${{ github.sha }}.zip"
        env:
          AWS_ACCESS_KEY_ID: ${{ steps.tf_output.outputs.ci_access_key_id }}
          AWS_SECRET_ACCESS_KEY: ${{ steps.tf_output.outputs.ci_secret_access_key }}
          AWS_REGION: ${{ env.AWS_REGION }}
```

Bagian workflow ini berfungsi untuk mengelola file penanda (pointer) versi artifact terbaru di bucket S3. Pertama, workflow mencoba mengunduh file latest.txt dari bucket S3. Jika file tersebut ada, maka file tersebut dibackup menjadi previous.txt di bucket yang sama, sehingga versi sebelumnya tetap tersimpan. Jika file tidak ditemukan, proses backup dilewati dan hanya menampilkan pesan informasi di log. Setelah itu, workflow membuat file latest.txt baru yang berisi nama file artifact zip hasil build terbaru (berdasarkan commit SHA), lalu meng-upload file ini ke bucket S3. Dengan cara ini, file latest.txt selalu menunjuk ke artifact build terbaru, dan versi sebelumnya tetap bisa diakses melalui previous.txt. Tujuan adanya diadakan ini adalah untuk membantu dalam fitur rollback sekiranya saat deploy gagal, maka dapat revert atau rollback ke previous version

Clean Up

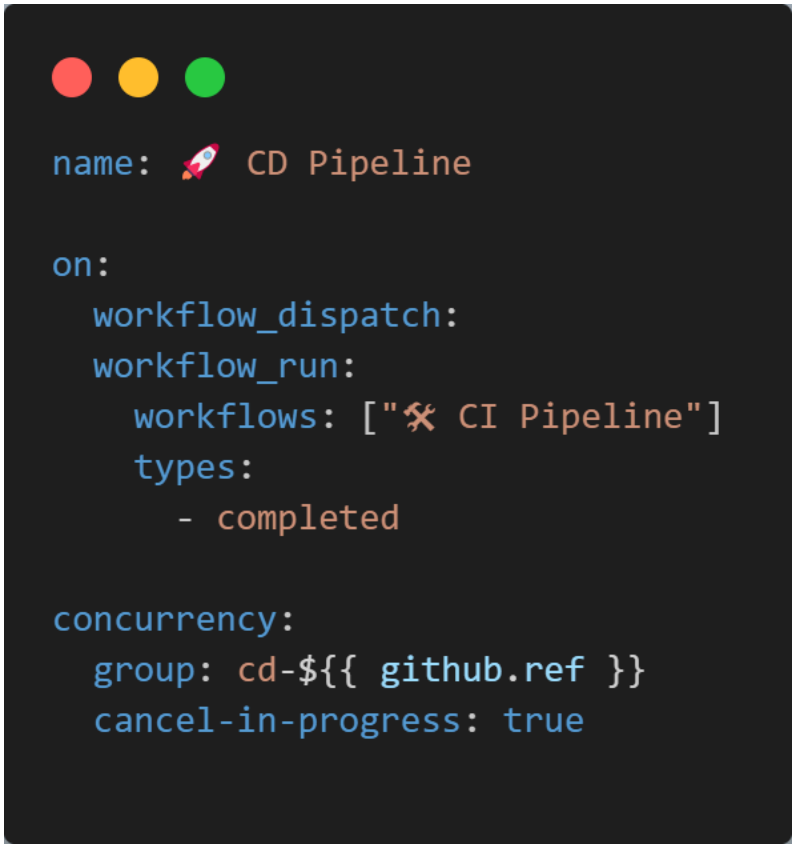


```
- name: 🧹 Clean Up Old Artifacts in S3
        run: |
          aws s3 cp s3://${{ steps.tf_output.outputs.artifact_bucket }}/latest.txt latest.txt || true
          aws s3 cp s3://${{ steps.tf_output.outputs.artifact_bucket }}/previous.txt previous.txt || true

          LATEST=$(cat latest.txt 2>/dev/null | xargs || echo "")
          PREVIOUS=$(cat previous.txt 2>/dev/null | xargs || echo "")

          echo "Keeping: [$LATEST] and [$PREVIOUS]"

          aws s3 ls s3://${{ steps.tf_output.outputs.artifact_bucket }}/ | awk '{print $4}' | grep '\.zip$' > all_zips.txt || true

          if [ -f all_zips.txt ]; then
            while read ZIPFILE; do
              if [[ -n "$ZIPFILE" && "$ZIPFILE" != "$LATEST" && "$ZIPFILE" != "$PREVIOUS" ]]; then
                echo "🗑️ Deleting $ZIPFILE"
                aws s3 rm s3://${{ steps.tf_output.outputs.artifact_bucket }}/$ZIPFILE
              else
                echo "✅ Keeping $ZIPFILE"
              fi
            done < all_zips.txt
          fi
        env:
          AWS_ACCESS_KEY_ID: ${{ steps.tf_output.outputs.ci_access_key_id }}
          AWS_SECRET_ACCESS_KEY: ${{ steps.tf_output.outputs.ci_secret_access_key }}
          AWS_REGION: ${{ env.AWS_REGION }}

      - name: 🧹 Clean artifacts and zip
        run: |
          rm -rf artifact artifact.zip
          mkdir artifact
```

Bagian workflow ini bertujuan untuk membersihkan file artifact lama di bucket S3 agar storage tetap efisien dan hanya menyimpan versi terbaru dan sebelumnya. Pertama, workflow mengunduh file latest.txt dan previous.txt dari bucket S3 untuk mengetahui nama artifact zip terbaru dan sebelumnya. Kemudian, workflow mengambil daftar semua file zip di bucket dan membandingkannya dengan nama file yang ingin dipertahankan. Jika ada file zip yang bukan merupakan versi terbaru atau sebelumnya, file tersebut akan dihapus dari bucket S3. Proses ini memastikan hanya dua versi artifact yang disimpan di S3, sehingga menghemat ruang dan menjaga kebersihan storage. Setelah proses di S3 selesai, workflow juga membersihkan folder dan file artifact lokal di runner untuk menjaga lingkungan build tetap bersih.

Penjelasan CD Pipeline

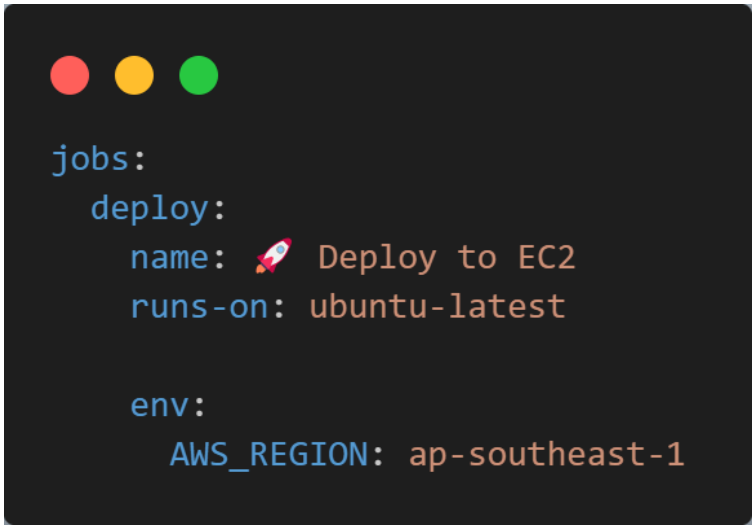
Inisialisasi CD Pipleline



```yaml
name: 🚀 CD Pipeline

on:
  workflow_dispatch:
  workflow_run:
    workflows: ["🛠️ CI Pipeline"]
    types:
      - completed

concurrency:
  group: cd-${{ github.ref }}
  cancel-in-progress: true
```

Bagian awal workflow ini mendefinisikan pipeline Continuous Deployment (CD) yang akan dijalankan secara otomatis setelah pipeline Continuous Integration (CI) selesai, atau bisa juga dijalankan manual melalui UI GitHub. Workflow ini menggunakan mekanisme concurrency untuk memastikan hanya satu proses CD yang berjalan untuk setiap branch pada satu waktu, dan jika ada proses baru yang dimulai, proses sebelumnya akan dibatalkan. Hal ini mencegah terjadinya deploy tumpang tindih dan menjaga agar deployment selalu konsisten dan up-to-date.

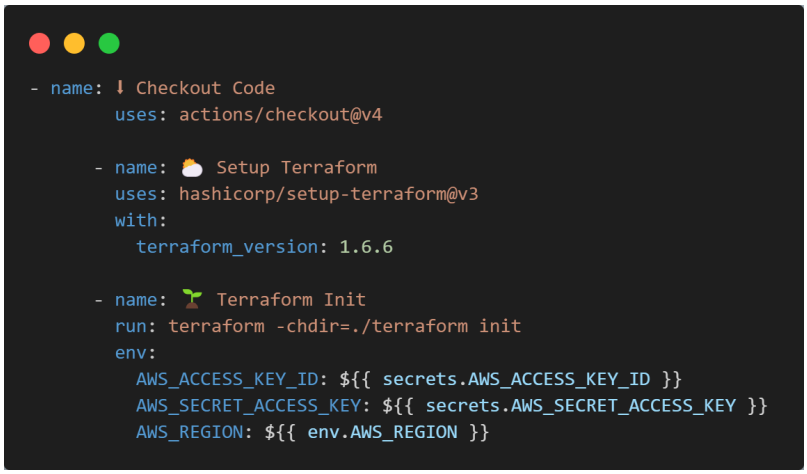
Inisialisasi CD Pipeline pt. 2



```yaml
jobs:
  deploy:
    name: 🚀 Deploy to EC2
    runs-on: ubuntu-latest

    env:
      AWS_REGION: ap-southeast-1
```

Bagian ini mendefinisikan sebuah job bernama deploy yang akan dijalankan pada runner GitHub dengan sistem operasi Ubuntu versi terbaru. Job ini bertujuan untuk melakukan proses deployment ke server EC2. Selain itu, environment variable AWS_REGION di-set ke ap-southeast-1 agar semua perintah yang berhubungan dengan AWS pada job ini menggunakan region Asia Tenggara 1.

Setup Terraform



```yaml
- name: ⬇ Checkout Code
      uses: actions/checkout@v4

    - name: 🐣 Setup Terraform
      uses: hashicorp/setup-terraform@v3
      with:
        terraform_version: 1.6.6

    - name: 🌱 Terraform Init
      run: terraform -chdir=./terraform init
      env:
        AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
        AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        AWS_REGION: ${{ env.AWS_REGION }}
```

Bagian workflow ini melakukan beberapa langkah awal untuk proses deployment. Pertama, kode repository diambil (checkout) ke runner menggunakan actions/checkout@v4. Selanjutnya, Terraform diinstal pada runner dengan versi 1.6.6 menggunakan hashicorp/setup-terraform@v3. Setelah itu, workflow menjalankan inisialisasi Terraform (terraform init) di direktori terraform dengan menggunakan kredensial AWS yang diambil dari secrets dan environment variable region yang

sudah di-set sebelumnya. Langkah-langkah ini memastikan environment sudah siap untuk proses deployment selanjutnya ke AWS.

Terraform Output

```yaml
- name: 📦 Terraform Output
        id: tf_output
        run: |
          terraform -chdir=./terraform output -json > tf_output.json
          echo "api_gateway_url=$(jq -r '.api_gateway_url.value' tf_output.json)" >> "$GITHUB_OUTPUT"
          echo "artifact_bucket=$(jq -r '.artifact_bucket_name.value' tf_output.json)" >> "$GITHUB_OUTPUT"
          echo "cd_access_key_id=$(jq -r '.cd_access_key_id.value' tf_output.json)" >> "$GITHUB_OUTPUT"
          echo "cd_secret_access_key=$(jq -r '.cd_secret_access_key.value' tf_output.json)" >> "$GITHUB_OUTPUT"
          echo "production_public_ip=$(jq -r '.production_ip.value' tf_output.json)" >> "$GITHUB_OUTPUT"
          echo "staging_public_ip=$(jq -r '.staging_ip.value' tf_output.json)" >> "$GITHUB_OUTPUT"
          echo "production_public_dns=$(jq -r '.production_public_dns.value' tf_output.json)" >> "$GITHUB_OUTPUT"
        env:
          AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
          AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          AWS_REGION: ${{ env.AWS_REGION }}

      - name: ✅ Check Terraform Output
        run: |
          echo "API Gateway URL: ${{ steps.tf_output.outputs.api_gateway_url }}"
          echo "Artifact Bucket: ${{ steps.tf_output.outputs.artifact_bucket }}"
          echo "Production Public IP: ${{ steps.tf_output.outputs.production_public_ip }}"
          echo "Staging Public IP: ${{ steps.tf_output.outputs.staging_public_ip }}"
```

Bagian workflow ini mengambil output dari Terraform yang sudah diinisialisasi sebelumnya. Output Terraform diambil dalam format JSON dan disimpan ke file tf_output.json. Selanjutnya, beberapa nilai penting seperti URL API Gateway, nama bucket artifact, kredensial akses CD, serta IP dan DNS publik untuk server staging dan production diekstrak dari file JSON tersebut dan disimpan ke environment output GitHub Actions agar bisa digunakan di langkah-langkah berikutnya. Setelah itu, workflow menampilkan nilai-nilai output utama ke log untuk memudahkan verifikasi hasil provisioning dan memastikan semua resource yang dibutuhkan untuk deployment sudah tersedia.

Download Latest Version

```yaml
- name: ☁️ Download latest.txt
        run: |
          aws s3 cp s3://${{ steps.tf_output.outputs.artifact_bucket }}/latest.txt ./latest.txt
        env:
          AWS_ACCESS_KEY_ID: ${{ steps.tf_output.outputs.cd_access_key_id }}
          AWS_SECRET_ACCESS_KEY: ${{ steps.tf_output.outputs.cd_secret_access_key }}
          AWS_REGION: ${{ env.AWS_REGION }}

      - name: 📦 Read version to deploy
        id: version
        run: |
          VERSION=$(cat latest.txt)
          echo "Deploying version: $VERSION"
          echo "version=$VERSION" >> $GITHUB_OUTPUT
```

Bagian workflow ini bertugas untuk mengambil informasi versi artifact yang akan dideploy. Pertama, file latest.txt diunduh dari bucket S3 menggunakan kredensial akses CD yang sudah diatur sebelumnya. File ini berisi nama file artifact zip hasil build terbaru. Setelah file berhasil diunduh, workflow membaca isi file tersebut untuk mendapatkan versi artifact yang akan dideploy, menampilkannya ke log, dan menyimpannya ke environment output GitHub Actions agar bisa digunakan pada langkah deployment berikutnya. Dengan cara ini, pipeline selalu menggunakan artifact versi terbaru yang sudah di-upload oleh proses CI.

Deploy to Staging

```yaml
- name: 📱 Deploy to EC2 Staging via SSH Script
        uses: appleboy/ssh-action@v1.0.3
        with:
          host: ${{ steps.tf_output.outputs.staging_public_ip }}
          username: ubuntu
          key: ${{ secrets.EC2_SSH_KEY }}
          script: |
            ARTIFACT_BUCKET="${{ steps.tf_output.outputs.artifact_bucket }}"
            AWS_REGION="${{ env.AWS_REGION }}"
            VERSION="${{ steps.version.outputs.version }}"

            # Create deploy folder first
            mkdir -p ~/deploy

            # Clean deploy folder
            rm -rf ~/deploy/*

            # Set ownership after folder exists
            sudo chown -R ubuntu:ubuntu ~/deploy

            # Download the ZIP artifact from S3
            aws s3 cp s3://$ARTIFACT_BUCKET/$VERSION ~/artifact.zip --region $AWS_REGION

            # Unzip it to the deploy folder
            unzip ~/artifact.zip -d ~/deploy

            # Navigate to the standalone directory
            cd ~/deploy/artifact/standalone

            # Set permissions
            sudo chmod +x server.js
            sudo chmod +x package.json

            # Restart PM2 with the new code
            pm2 restart book-library || pm2 start server.js --name book-library
            pm2 save
```

Bagian workflow ini menjalankan proses deployment otomatis ke server EC2 staging menggunakan SSH. Dengan menggunakan action appleboy/ssh-action, pipeline terhubung ke server EC2 menggunakan IP publik, username, dan private key yang sudah diatur di secrets. Di dalam server, pipeline membuat dan membersihkan folder ~/deploy, lalu mengatur kepemilikan folder agar bisa diakses user ubuntu. Artifact hasil build terbaru diunduh dari bucket S3 sesuai versi yang sudah ditentukan, lalu diekstrak ke folder deploy. Setelah itu, pipeline masuk ke direktori

hasil ekstrak, mengatur permission file penting, dan menjalankan atau me-restart aplikasi menggunakan PM2 agar aplikasi berjalan dengan kode terbaru. Proses ini memastikan deployment ke staging selalu menggunakan artifact build terbaru dari pipeline CI.

Smoke Test Staging

```yaml
- name: 🧪 Run Playwright Smoke Tests (UI, Dark Mode, Search)
        id: staging_smoke_tests
        continue-on-error: true
        run: |
          echo "Waiting for server to be ready..."
          sleep 15

          # Simpan HTTP status code ke dalam variabel
          STATUS_CODE=$(curl -s -o /dev/null -w "%{http_code}" ${{ steps.tf_output.outputs.api_gateway_url }}/health)

          # Periksa apakah status code bukan 200
          if [ "$STATUS_CODE" -ne 200 ]; then
            echo "Health check failed with status code: $STATUS_CODE"
            exit 1 # Keluar dengan error, step ini akan gagal
          else
            echo "Health check successful with status 200. Proceeding with tests."
          fi

          npm ci
          npx playwright install --with-deps
          npm run smoke
        env:
          SMOKE_UI_URL: http://${{ steps.tf_output.outputs.staging_public_ip }}:3000
          AWS_API_URL: ${{ steps.tf_output.outputs.api_gateway_url }}
```

Setelah dideploy pada staging, langkah selanjutnya adalah melakukan smoke test ui menggunakan playwright serta health test. Langkah pertama adalah untuk mengenvoke /health pada api. /health ini hanya mengembalikan status 200 atau ok dan akan dicek pada if statement tersebut, misalkan tidak 200 maka health check failed, yang berarti api url belum dapat digunakan. Jika sudah aman, berarti langsung ke playwright test yang melakukan testing UI langsung dengan mensimulasikan browser dan interactions. Jika playwright sudah berhasil, maka bisa langsung ke step berikutnya. Jika gagal, maka akan dilakukan roll back, dan tidak berhasil deploy ke production.

Deploy to Production

```yaml
- name: 📱 Deploy to EC2 via SSH Script
        if: steps.staging_smoke_tests.outcome == 'success'
        uses: appleboy/ssh-action@v1.0.3
        with:
          host: ${{ steps.tf_output.outputs.production_public_ip }}
          username: ubuntu
          key: ${{ secrets.EC2_SSH_KEY }}
          script: |
            ARTIFACT_BUCKET="${{ steps.tf_output.outputs.artifact_bucket }}"
            AWS_REGION="${{ env.AWS_REGION }}"
            VERSION="${{ steps.version.outputs.version }}"

            # Stop running process
            pm2 stop book-library || true

            # Create deploy folder first
            mkdir -p ~/deploy

            # Clean deploy folder and artifact
            rm -rf ~/deploy/*
            rm -f ~/artifact.zip

            # Set ownership after folder exists
            sudo chown -R ubuntu:ubuntu ~/deploy

            # Download the ZIP artifact from S3
            aws s3 cp s3://$ARTIFACT_BUCKET/$VERSION ~/artifact.zip --region $AWS_REGION

            # Unzip it to the deploy folder
            unzip ~/artifact.zip -d ~/deploy

            # Navigate to the standalone directory
            cd ~/deploy/artifact/standalone

            # Set permissions
            sudo chmod +x server.js

            # Wait a moment before starting
            sleep 5

            # Restart PM2 with the new code
            pm2 restart book-library --update-env || pm2 start server.js --name book-library
            pm2 save
```

Bila smoke test pada staging berhasil, maka kita akan langsung deploy hasil stagiing tersebut ke production. Kurang lebih untuk steps deploy tidak terlalu jauh berbeda dengan steps pada deploy staging. Setelah deploy di production, akan

dilakukan smoke test lagi dengan isi yang sama seperti smoke test sebelumnya untuk memastikan fungsional website pasti bisa digunakan.

Print out summary

```
- name: 📝 Add deployment info to summary
        if: steps.production_smoke_tests.outcome == 'success'
        run: |
          echo "### ✅ Deployed successfully!" >> $GITHUB_STEP_SUMMARY
          echo "**Latest commit:** \`$(echo $GITHUB_SHA | cut -c1-7)\`" >> $GITHUB_STEP_SUMMARY
          echo "**Preview URL:** [${{ steps.tf_output.outputs.production_public_dns }}:3000](${{ steps.tf_output.outputs.production_public_dns }}:3000)" >> $GITHUB_STEP_SUMMARY
```

Bila semuanya sudah berhasil, maka akan diprint bahwa deploy succesful serta priview URL nya

Rollback

```yaml
- name: 📄 Rollback to Previous Success Version
        if: steps.staging_smoke_tests.outcome == 'failure'
        run: |
          aws s3 cp s3://${{ steps.tf_output.outputs.artifact_bucket }}/previous.txt previous.txt

          VERSION=$(cat previous.txt)

          if [ -z "$VERSION" ]; then
            echo "::error::Failed to read version from previous.txt. The file might be empty."
            exit 1
          fi

          echo "Rolling back latest.txt to version: $VERSION"
          echo "$VERSION" > latest.txt
          aws s3 cp latest.txt s3://${{ steps.tf_output.outputs.artifact_bucket }}/latest.txt

          echo "✅ Rolled back: latest.txt now points to $VERSION"
        env:
          AWS_ACCESS_KEY_ID: ${{ steps.tf_output.outputs.cd_access_key_id }}
          AWS_SECRET_ACCESS_KEY: ${{ steps.tf_output.outputs.cd_secret_access_key }}
          AWS_REGION: ${{ env.AWS_REGION }}
```

Jika tidak berhasi smoke test di staging, maka ia akan rollback menggunakan previous.txt dan mengganti version yang previous tersebut ke latest.

Stop Deployment in Staging

```yaml
- name: 🗒 Stop Deployment in Staging EC2
        if: always()
        uses: appleboy/ssh-action@v1.0.3
        with:
          host: ${{ steps.tf_output.outputs.staging_public_ip }}
          username: ubuntu
          key: ${{ secrets.EC2_SSH_KEY }}
          script: |
            # Stop Running pm2 Process
            pm2 stop book-library || true

            # Clean deploy folder
            rm -rf ~/deploy
```

Setelah semua testing dilakukan dan antara deployment ke production berhasil ataupun tidak, deployment di staging otomatis dihapus, agar tidak tidak menghabiskan resource.