Memulai dengan Terraform: Panduan untuk dan para ahli AWS CDK AWS CloudFormation

AWS Panduan Preskriptif

AWS Panduan Preskriptif: Memulai dengan Terraform: Panduan untuk dan para ahli AWS CDK AWS CloudFormation

Table of Contents

Memulai dengan Terraform: Panduan untuk ahli AWS CDK dan AWS CloudFormation

Steven Guggenheimer, Amazon Web Services ()AWS

Maret 2024 (sejarah dokumen)

Jika pengalaman Anda dengan penyediaan sumber daya cloud secara eksklusif berada dalam ranah AWS, Anda mungkin memiliki pengalaman terbatas dengan alat infrastruktur sebagai kode (IAc) di luar dan. AWS Cloud Development Kit (AWS CDK)AWS CloudFormation Bahkan, alat serupa, seperti Hashicorp Terraform, mungkin sama sekali tidak Anda kenal. Namun, semakin dalam Anda memasuki perjalanan cloud Anda, semakin tak terhindarkan Anda akan menemukan Terraform. Ini akan jelas menguntungkan Anda untuk terbiasa dengan konsep intinya.

Sementara Terraform, yang AWS CDK, dan CloudFormation mencapai tujuan yang sama dan berbagi banyak konsep inti, ada beberapa perbedaan. Anda mungkin tidak siap untuk perbedaan ini jika Anda mendekati Terraform untuk pertama kalinya. Bagaimanapun, AWS CDK dan CloudFormation tumpukan semuanya didasarkan di dalam Akun AWS, jadi dengan cara itu, mereka memiliki hubungan langsung dengan sebagian besar sumber daya yang mereka pertahankan. Terraform tidak berbasis dalam lingkungan penyedia cloud tunggal mana pun. Ini memberikan fleksibilitas untuk mendukung berbagai penyedia yang berbeda, tetapi harus mempertahankan sumber daya dari jumlah ke lokasi terpencil.

Panduan ini membantu mengungkap konsep inti di balik Terraform untuk membantu Anda menangani tantangan IAc apa pun yang menghampiri Anda. Ini berfokus pada bagaimana Terraform menggunakan konsep, seperti penyedia, modul, dan file negara, untuk menyediakan sumber daya. Ini juga kontras konsep Terraform dengan bagaimana AWS CDK dan CloudFormation melakukan operasi serupa.

> **Note**
>
> AWS CDK Ini membantu pengembang menyebarkan CloudFormation tumpukan dengan menggunakan bahasa pengkodean terprogram. Setelah Anda menjalankan`cdk synth`, kode Anda diubah menjadi CloudFormation template. Sejak saat itu, prosesnya identik antara AWS CDK dan CloudFormation. Demi singkatnya, panduan ini biasanya mengacu pada proses AWS IAc dalam CloudFormation istilah, tetapi perbandingannya sama tepat untuk. AWS CDK

CloudFormation dan terminologi Terraform

Ketika membandingkan Terraform dengan AWS CDK dan CloudFormation, merekonsiliasi konsep inti IAc bisa jadi sulit karena terminologi yang tidak konsisten yang digunakan untuk menggambarkannya. Berikut ini adalah istilah-istilah ini dan bagaimana panduan ini akan merujuknya:

- Stack — Tumpukan adalah IAc yang digunakan ke dalam pipa CI/CD dan dapat dilacak sebagai satu unit.Meskipun istilah ini umum di CloudFormation, Terraform tidak benar-benar menggunakan istilah ini. Tumpukan Terraform adalah modul root yang diterapkan dengan semua modul turunannya. Namun, untuk menghindari kebingungan dengan istilah modul, panduan ini menggunakan istilah stack untuk menggambarkan penerapan tunggal untuk kedua alat.
- Status - Status adalah semua sumber daya yang saat ini dilacak dan konfigurasinya saat ini dalam tumpukan penyebaran IAC. Seperti yang dijelaskan di Memahami status dan backend Terraform bagian, Terraform menggunakan istilah status lebih dari. CloudFormation Ini karena mempertahankan status lebih terlihat di Terraform, tetapi melacak dan memperbarui status sama pentingnya. CloudFormation
- File IAC - File IAC adalah file tunggal yang berisi infrastruktur sebagai bahasa kode (IAc). CloudFormation mengacu pada satu CloudFormation file sebagai template. Namun template dan file template di Terraform adalah sesuatu yang sama sekali berbeda. Setara dengan CloudFormation template di Terraform disebut file konfigurasi. Untuk meminimalkan kebingungan dalam panduan ini, file istilah atau file IAC digunakan untuk merujuk ke CloudFormation templat dan file konfigurasi Terraform.

Tabel berikut membandingkan terminologi yang digunakan untuk CloudFormation dan Terraform. Maksud dari tabel ini adalah untuk menunjukkan kesamaan. Ini bukan one-to-one perbandingan. Setiap konsep berbeda setidaknya sedikit antara CloudFormation dan Terraform. Konsep dijelaskan secara mendalam di bagian yang relevan dari panduan ini.

CloudFormation istilah	Istilah Terraform	Bagian dari panduan ini
Antarmuka CDK (seperti iBucket)	Sumber data	Memahami sumber data Terraform
Ubah set	Rencana	Memahami modul Terraform

CloudFormation istilah	Istilah Terraform	Bagian dari panduan ini
Fungsi syarat	Ekspresi bersyarat	Memahami fungsi, ekspresi, dan meta-argumen Terraform
`DependsOn` atribut	`depends_on` meta-argumen	Memahami fungsi, ekspresi, dan meta-argumen Terraform
Fungsi intrinsik	Fungsi	Memahami fungsi, ekspresi, dan meta-argumen Terraform
Modul	Modul	Memahami modul Terraform
Output	Nilai keluaran	Memahami variabel Terraform, nilai lokal, dan output
Parameter	Variabel	Memahami variabel Terraform, nilai lokal, dan output
Registri	Penyedia	Memahami penyedia Terraform
Templat	File konfigurasi	Semua

Memahami sumber daya Terraform

Alasan utama keberadaan keduanya AWS CloudFormation dan Terraform adalah pembuatan dan pemeliharaan sumber daya cloud. Tapi apa sebenarnya sumber daya cloud itu? Dan apakah CloudFormation sumber daya dan sumber daya Terraform adalah hal yang sama? Jawabannya adalah... ya dan tidak. Untuk mengilustrasikan hal ini, panduan ini memberikan contoh penggunaan CloudFormation dan kemudian Terraform untuk membuat bucket Amazon Simple Storage Service (Amazon S3).

Contoh CloudFormation kode berikut membuat contoh bucket Amazon S3.

```
{
    "myS3Bucket": {
        "Type": "AWS::S3::Bucket",
        "Properties": {
            "BucketName": "my-s3-bucket",
            "BucketEncryption": {
                "ServerSideEncryptionConfiguration": [
                    {
                        "ServerSideEncryptionByDefault": {
                            "SSEAlgorithm": "AES256"
                        }
                    }
                ]
            },
            "PublicAccessBlockConfiguration": {
                "BlockPublicAcls": true,
                "BlockPublicPolicy": true,
                "IgnorePublicAcls": true,
                "RestrictPublicBuckets": true
            },
            "VersioningConfiguration": {
                "Status": "Enabled"
            }
        }
    }
}
```

Contoh kode Terraform berikut membuat bucket Amazon S3 yang identik.

```
resource "aws_s3_bucket" "myS3Bucket" {
```

```
  bucket = "my-s3-bucket"
}

resource "aws_s3_bucket_server_side_encryption_configuration" "bucketencryption" {
  bucket = aws_s3_bucket.myS3Bucket.id
  rule {
    apply_server_side_encryption_by_default {
      sse_algorithm = "AES256"
    }
  }
}

resource "aws_s3_bucket_public_access_block" "publicaccess" {
  bucket                  = aws_s3_bucket.myS3Bucket.id
  block_public_acls       = true
  block_public_policy     = true
  ignore_public_acls      = true
  restrict_public_buckets = true
}

resource "aws_s3_bucket_versioning" "versioning" {
  bucket = aws_s3_bucket.myS3Bucket.id
  versioning_configuration {
    status = "Enabled"
  }
}
```

Untuk Terraform, penyedia mendefinisikan sumber daya, lalu pengembang mendeklarasikan dan mengonfigurasi sumber daya tersebut. Penyedia adalah konsep yang dibahas panduan ini di bagian selanjutnya. Contoh Terraform membuat sumber daya yang sepenuhnya terpisah untuk beberapa pengaturan bucket S3. Membuat sumber daya terpisah untuk pengaturan belum tentu tipikal cara AWS Penyedia Terraform memperlakukan AWS sumber daya. Namun, contoh ini menunjukkan perbedaan penting. Sementara CloudFormation sumber daya ditentukan secara ketat oleh spesifikasi CloudFormation sumber daya, Terraform tidak memiliki persyaratan seperti itu. Di Terraform, konsep sumber daya sedikit lebih samar-samar.

Meskipun alat mungkin berbeda mengenai pagar pembatas yang tepat yang menentukan apa itu sumber daya tunggal, secara umum, sumber daya cloud adalah entitas tertentu yang ada di cloud dan yang dapat dibuat, diperbarui, atau dihapus. Jadi terlepas dari berapa banyak sumber daya yang terlibat, dua contoh sebelumnya keduanya membuat hal yang sama persis dengan pengaturan yang sama persis dalam file Akun AWS.

Memahami penyedia Terraform

Di Terraform, penyedia adalah plugin yang berinteraksi dengan penyedia cloud, alat pihak ketiga, dan API lainnya. Untuk menggunakan Terraform dengan AWS, Anda menggunakan AWS Penyedia, yang berinteraksi dengan sumber daya. AWS

Jika Anda belum pernah menggunakan AWS CloudFormation registri untuk memasukkan ekstensi pihak ketiga ke dalam tumpukan penerapan Anda, maka penyedia Terraform mungkin perlu membiasakan diri. Karena CloudFormation asli AWS, penyedia AWS sumber daya sudah ada secara default. Terraform, di sisi lain, tidak memiliki penyedia default tunggal, jadi tidak ada yang dapat diasumsikan tentang asal-usul sumber daya tertentu. Ini berarti bahwa hal pertama yang perlu dideklarasikan dalam file konfigurasi Terraform adalah persis ke mana sumber daya pergi dan bagaimana mereka akan sampai di sana.

Perbedaan ini menambahkan lapisan kompleksitas ekstra ke Terraform yang tidak ada dengannya. CloudFormation Namun, kompleksitas itu memberikan peningkatan fleksibilitas. Anda dapat mendeklarasikan beberapa penyedia dalam satu modul Terraform, dan kemudian sumber daya dasar yang dibuat dapat berinteraksi satu sama lain sebagai bagian dari lapisan penerapan yang sama.

Ini bisa berguna dalam berbagai cara. Penyedia tidak harus untuk penyedia cloud terpisah. Penyedia dapat mewakili sumber apa pun untuk sumber daya cloud. Misalnya, ambil Amazon Elastic Kubernetes Service (Amazon EKS). Saat Anda menyediakan klaster Amazon EKS, Anda mungkin ingin menggunakan bagan Helm untuk mengelola ekstensi pihak ketiga dan menggunakan Kubernetes sendiri untuk mengelola sumber daya pod. Karena AWS, Helm, dan Kubernetes semuanya memiliki penyedia Terraform sendiri, Anda dapat menyediakan dan mengintegrasikan semua sumber daya ini secara bersamaan dan kemudian meneruskan nilai di antara mereka.

Dalam contoh kode berikut untuk Terraform, AWS Provider membuat cluster Amazon EKS, dan kemudian informasi konfigurasi Kubernetes yang dihasilkan diteruskan ke penyedia Helm dan Kubernetes.

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = ">= 4.33.0"
    }

    helm = {
```

```
      source  = "hashicorp/helm"
      version = "2.12.1"
    }

    kubernetes = {
      source  = "hashicorp/kubernetes"
      version = "2.26.0"
    }
  }
  required_version = ">= 1.2.0"
}

provider "aws" {
  region = "us-west-2"
}

resource "aws_eks_cluster" "example_0" {
  name     = "example_0"
  role_arn = aws_iam_role.cluster_role.arn
  vpc_config {
    endpoint_private_access = true
    endpoint_public_access  = true
    subnet_ids              = var.subnet_ids
  }
}

locals {
  host        = aws_eks_cluster.example_0.endpoint
  certificate = base64decode(aws_eks_cluster.example_0.certificate_authority.data)
}

provider "helm" {
  kubernetes {
    host                   = local.host
    cluster_ca_certificate = local.certificate
    # exec allows for an authentication command to be run to obtain user
    # credentials rather than having them stored directly in the file
    exec {
      api_version = "client.authentication.k8s.io/v1beta1"
      args        = ["eks", "get-token", "--cluster-name",
 aws_eks_cluster.example_0.name]
      command     = "aws"
    }
```

```
  }
}

provider "kubernetes" {
  host                   = local.host
  cluster_ca_certificate = local.certificate
  exec {
    api_version = "client.authentication.k8s.io/v1beta1"
    args        = ["eks", "get-token", "--cluster-name",
 aws_eks_cluster.example_0.name]
    command     = "aws"
  }
}
```

Ada trade-off mengenai penyedia ketika datang ke dua alat IAc. Terraform sepenuhnya bergantung pada paket penyedia yang terletak di luar, yang merupakan mesin yang mendorong penerapannya. CloudFormation secara internal mendukung semua AWS proses utama. Dengan CloudFormation, Anda perlu khawatir tentang penyedia pihak ketiga hanya jika Anda ingin memasukkan ekstensi pihak ketiga. Ada pro dan kontra untuk setiap pendekatan. Mana yang tepat untuk Anda berada di luar cakupan panduan ini, tetapi penting untuk mengingat perbedaannya saat mengevaluasi kedua alat.

Menggunakan alias Terraform

Di Terraform, Anda dapat meneruskan konfigurasi khusus ke setiap penyedia. Jadi bagaimana jika Anda ingin menggunakan beberapa konfigurasi penyedia dalam modul yang sama? Dalam hal ini Anda harus menggunakan alias. Alias membantu Anda memilih penyedia mana yang akan digunakan pada tingkat per sumber daya atau per modul. Bila Anda memiliki lebih dari satu instance dari penyedia yang sama, Anda menggunakan alias untuk menentukan instance non-default. Misalnya, instance penyedia default Anda mungkin spesifik Wilayah AWS, tetapi Anda menggunakan alias untuk menentukan wilayah alternatif.

Contoh Terraform berikut menunjukkan cara menggunakan alias untuk menyediakan bucket di berbeda. Wilayah AWS Wilayah default untuk penyedia adalah`us-west-2`, tetapi Anda dapat menggunakan alias timur untuk menyediakan sumber daya di`us-east-2`.

```
provider "aws" {
  region = "us-west-2"
}
```

```
provider "aws" {
  alias  = "east"
  region = "us-east-2"
}

resource "aws_s3_bucket" "myWestS3Bucket" {
  bucket = "my-west-s3-bucket"
}

resource "aws_s3_bucket" "myEastS3Bucket" {
  provider = aws.east
  bucket   = "my-east-s3-bucket"
}
```

Bila Anda menggunakan `alias` bersama dengan `provider` meta-argumen, seperti yang ditunjukkan pada contoh sebelumnya, Anda dapat menentukan konfigurasi penyedia yang berbeda untuk sumber daya tertentu. Penyediaan sumber daya dalam beberapa Wilayah AWS dalam satu tumpukan hanyalah permulaan. Penyedia aliasing sangat nyaman dalam banyak hal.

Misalnya, sangat umum untuk menyediakan beberapa cluster Kubernetes sekaligus. Alias dapat membantu Anda mengonfigurasi penyedia Helm dan Kubernetes tambahan sehingga Anda dapat menggunakan alat pihak ketiga ini secara berbeda untuk berbagai sumber daya Amazon EKS. Contoh kode Terraform berikut menggambarkan cara menggunakan alias untuk melakukan tugas ini.

```
resource "aws_eks_cluster" "example_0" {
  name     = "example_0"
  role_arn = aws_iam_role.cluster_role.arn
  vpc_config {
    endpoint_private_access = true
    endpoint_public_access  = true
    subnet_ids              = var.subnet_ids[0]
  }
}

resource "aws_eks_cluster" "example_1" {
  name     = "example_1"
  role_arn = aws_iam_role.cluster_role.arn
  vpc_config {
    endpoint_private_access = true
    endpoint_public_access  = true
    subnet_ids              = var.subnet_ids[1]
  }
```

```
}

locals {
  host         = aws_eks_cluster.example_0.endpoint
  certificate  = base64decode(aws_eks_cluster.example_0.certificate_authority.data)
  host1        = aws_eks_cluster.example_1.endpoint
  certificate1 = base64decode(aws_eks_cluster.example_1.certificate_authority.data)
}

provider "helm" {
  kubernetes {
    host                   = local.host
    cluster_ca_certificate = local.certificate
    exec {
      api_version = "client.authentication.k8s.io/v1beta1"
      args        = ["eks", "get-token", "--cluster-name",
 aws_eks_cluster.example_0.name]
      command     = "aws"
    }
  }
}

provider "helm" {
  alias = "helm1"
  kubernetes {
    host                   = local.host1
    cluster_ca_certificate = local.certificate1
    exec {
      api_version = "client.authentication.k8s.io/v1beta1"
      args        = ["eks", "get-token", "--cluster-name",
 aws_eks_cluster.example_1.name]
      command     = "aws"
    }
  }
}

provider "kubernetes" {
  host                   = local.host
  cluster_ca_certificate = local.certificate
  exec {
    api_version = "client.authentication.k8s.io/v1beta1"
    args        = ["eks", "get-token", "--cluster-name",
 aws_eks_cluster.example_0.name]
```

```
    command     = "aws"
  }
}

provider "kubernetes" {
  alias                  = "kubernetes1"
  host                   = local.host1
  cluster_ca_certificate = local.certificate1
  exec {
    api_version = "client.authentication.k8s.io/v1beta1"
    args        = ["eks", "get-token", "--cluster-name",
 aws_eks_cluster.example_1.name]
    command     = "aws"
  }
}
```

Memahami modul Terraform

Dalam bidang infrastruktur sebagai kode (IAc), modul adalah blok kode mandiri yang diisolasi dan dikemas bersama untuk digunakan kembali. Konsep modul adalah aspek yang tak terhindarkan dari pengembangan Terraform. Untuk informasi selengkapnya, lihat Modul dalam dokumentasi Terraform. AWS CloudFormation juga mendukung modul. Untuk informasi selengkapnya, lihat Memperkenalkan AWS CloudFormation modul di Blog Operasi dan Migrasi AWS Cloud.

Perbedaan utama antara modul di Terraform dan CloudFormation modul CloudFormation diimpor dengan menggunakan tipe sumber daya khusus ()`AWS::CloudFormation::ModuleVersion`. Di Terraform, setiap konfigurasi memiliki setidaknya satu modul, yang dikenal sebagai modul root. Sumber daya Terraform yang ada di file.tf utama atau file dalam file konfigurasi Terraform dianggap ada di modul root. Modul root kemudian dapat memanggil modul lain untuk dimasukkan dalam tumpukan. Contoh berikut menunjukkan modul root yang menyediakan cluster Amazon Elastic Kubernetes Service (Amazon EKS) dengan menggunakan modul eks open source.

```
terraform {
  required_providers {
    helm = {
      source  = "hashicorp/helm"
      version = "2.12.1"
    }
  }
  required_version = ">= 1.2.0"
}

module "eks" {
  source  = "terraform-aws-modules/eks/aws"
  version = "20.2.1"
  vpc_id  = var.vpc_id
}

provider "helm" {
  kubernetes {
    host                   = module.eks.cluster_endpoint
    cluster_ca_certificate =
 base64decode(module.eks.cluster_certificate_authority_data)
  }
}
```

Anda mungkin telah memperhatikan bahwa file konfigurasi di atas tidak menyertakan AWS Penyedia. Itu karena modul mandiri dan dapat mencakup penyedia mereka sendiri. Karena penyedia Terraform bersifat global, penyedia dari modul anak dapat digunakan dalam modul root. Ini tidak benar tentang semua nilai modul. Nilai internal lain dalam modul dicakup secara default ke modul itu saja dan perlu dideklarasikan sebagai output agar dapat diakses di modul root. Anda dapat memanfaatkan modul open source untuk menyederhanakan pembuatan sumber daya dalam tumpukan Anda. Misalnya, modul eks melakukan lebih dari sekedar menyediakan kluster EKS — modul ini menyediakan lingkungan Kubernetes yang berfungsi penuh. Menggunakannya dapat menyelamatkan Anda dari menulis lusinan baris kode tambahan, asalkan konfigurasi modul eks sesuai dengan kebutuhan Anda.

Memanggil modul

Dua perintah CLI Terraform utama yang Anda jalankan selama penerapan Terraform adalah terraform init dan terraform apply. Salah satu langkah default yang dilakukan `terraform init` perintah adalah menemukan semua modul anak dan mengimpornya sebagai dependensi ke dalam direktori. `.terraform/modules` Selama pengembangan, setiap kali Anda menambahkan modul bersumber eksternal baru, Anda harus menginisialisasi ulang sebelum menggunakan perintah. `apply` Saat Anda mendengar referensi ke modul Terraform, itu mengacu pada paket di direktori ini. Sebenarnya, modul yang Anda deklarasikan dalam kode Anda adalah modul panggilan, jadi dalam praktiknya, kata kunci modul memanggil modul yang sebenarnya, yang disimpan sebagai dependensi.

Dengan cara ini, modul panggilan berfungsi sebagai perwakilan yang lebih ringkas dari modul lengkap yang akan diganti saat penerapan berlangsung. Anda dapat memanfaatkan ide ini dengan membuat modul Anda sendiri di dalam tumpukan Anda untuk menegakkan pemisahan logis sumber daya dengan menggunakan kriteria apa pun yang Anda inginkan. Ingatlah bahwa tujuan akhir dari melakukan ini adalah untuk mengurangi kompleksitas tumpukan Anda. Karena berbagi data antar modul mengharuskan Anda untuk mengeluarkan data itu dari dalam modul, terkadang terlalu mengandalkan modul dapat memperumit banyak hal.

Modul root

Karena setiap konfigurasi Terraform memiliki setidaknya satu modul, ini dapat membantu untuk memeriksa properti modul dari modul yang paling sering Anda tangani: modul root. Setiap kali Anda mengerjakan proyek Terraform, modul root terdiri dari semua `.tf` (atau`.tf.json`) file di direktori tingkat atas Anda. Saat Anda menjalankan `terraform apply` direktori tingkat atas

itu, Terraform mencoba menjalankan setiap `.tf` file yang ditemukannya di sana. Setiap file di subdirektori diabaikan kecuali mereka dipanggil dalam salah satu file konfigurasi tingkat atas ini.

Ini memberikan beberapa fleksibilitas dalam cara Anda menyusun kode Anda. Ini juga merupakan alasan mengapa lebih akurat untuk merujuk pada penerapan Terraform Anda sebagai modul daripada sebagai file karena beberapa file dapat terlibat dalam satu proses. Ada struktur modul standar yang direkomendasikan Terraform untuk praktik terbaik. Namun, jika Anda meletakkan `.tf` file apa pun di direktori tingkat atas Anda, itu akan berjalan bersama dengan file lainnya. Faktanya, semua `.tf` file tingkat atas dalam modul digunakan saat Anda menjalankannya. `terraform apply` Jadi file mana yang dijalankan Terraform lebih dulu? Jawaban atas pertanyaan itu sangat penting.

Ada serangkaian langkah yang dilakukan Terraform setelah inisialisasi dan sebelum penerapan tumpukan. Pertama, konfigurasi yang ada dianalisis, dan kemudian grafik ketergantungan dibuat. Grafik ketergantungan menentukan sumber daya apa yang diminta dan dalam urutan apa mereka harus ditangani. Sumber daya yang berisi properti yang direferensikan dalam sumber daya lain, misalnya, akan ditangani sebelum sumber daya dependennya. Demikian pula, sumber daya yang secara eksplisit menyatakan ketergantungan dengan menggunakan `depends_on` parameter akan ditangani setelah sumber daya yang mereka tentukan. Jika memungkinkan, Terraform dapat menerapkan paralelisme dan menangani sumber daya yang tidak bergantung secara bersamaan. Anda dapat melihat grafik ketergantungan sebelum menerapkan dengan menggunakan perintah grafik terraform.

Setelah grafik ketergantungan dibuat, Terraform menentukan apa yang perlu dilakukan selama penerapan. Ini membandingkan grafik ketergantungan dengan file status terbaru. Hasil dari proses ini disebut rencana, dan ini sangat mirip dengan set CloudFormation perubahan. Anda dapat melihat paket saat ini dengan menggunakan perintah terraform plan.

Sebagai praktik terbaik, disarankan untuk tetap sedekat mungkin dengan struktur modul standar. Dalam kasus di mana file konfigurasi Anda menjadi terlalu panjang untuk dikelola secara efisien dan pemisahan logis dapat menyederhanakan manajemen, Anda dapat menyebarkan kode Anda di beberapa file. Ingatlah bagaimana grafik ketergantungan dan proses rencana bekerja untuk membuat tumpukan Anda berjalan seefisien mungkin.

Memahami status dan backend Terraform

Salah satu konsep terpenting dalam infrastruktur sebagai kode (IAc) adalah konsep negara. Layanan IAc mempertahankan status, yang memungkinkan Anda mendeklarasikan sumber daya dalam file IAc tanpa membuatnya ulang setiap kali Anda menerapkan. File IAC mendokumentasikan status semua sumber daya pada akhir penerapan sehingga kemudian dapat membandingkan status itu dengan status target, seperti yang dinyatakan dalam penerapan berikutnya. Jadi, jika status saat ini berisi bucket Amazon Simple Storage Service (Amazon S3) `my-s3-bucket` bernama Simple Storage Service (Amazon S3) dan perubahan yang masuk juga berisi bucket yang sama, proses baru akan menerapkan perubahan apa pun yang ditemukan pada bucket yang ada daripada mencoba membuat bucket baru.

Tabel berikut memberikan contoh proses status IAc umum.

Keadaan saat ini	Status target	Tindakan
Tidak ada ember S3 bernama `my-s3-bucket`	Ember S3 bernama `my-s3-bucket`	Buat bucket S3 bernama `my-s3-bucket`
`my-s3-bucket` tanpa pembuatan versi bucket yang dikonfigurasi	`my-s3-bucket` tanpa pembuatan versi bucket yang dikonfigurasi	Tidak ada tindakan
`my-s3-bucket` tanpa pembuatan versi bucket yang dikonfigurasi	`my-s3-bucket` dengan pembuatan versi bucket yang dikonfigurasi	`my-s3-bucket` Konfigura sikan untuk memiliki versi bucket
`my-s3-bucket` dengan pembuatan versi bucket yang dikonfigurasi	Tidak ada ember S3 bernama `my-s3-bucket`	Mencoba untuk menghapus `my-s3-bucket`

Untuk memahami berbagai cara di mana AWS CloudFormation dan status trek Terraform, penting untuk mengingat perbedaan dasar pertama antara kedua alat: CloudFormation di-host di dalam AWS Cloud, dan Terraform pada dasarnya jarak jauh. Fakta ini memungkinkan CloudFormation untuk mempertahankan keadaan secara internal. Anda dapat pergi ke CloudFormation konsol dan melihat

riwayat peristiwa dari tumpukan tertentu, tetapi CloudFormation layanan itu sendiri memberlakukan aturan status untuk Anda.

Tiga mode yang CloudFormation beroperasi di bawah untuk sumber daya tertentu adalah`Create`,`Update`, dan`Delete`. Mode saat ini ditentukan berdasarkan apa yang terjadi pada penerapan terakhir, dan tidak dapat dipengaruhi sebaliknya. Anda mungkin dapat memperbarui CloudFormation sumber daya secara manual untuk memengaruhi mode mana yang ditentukan, tetapi Anda tidak dapat meneruskan perintah CloudFormation yang mengatakan “Untuk sumber daya ini, operasikan dalam `Create` mode.”

Karena Terraform tidak di-host di AWS Cloud, proses pemeliharaan status harus lebih dapat dikonfigurasi. Untuk alasan ini, status Terraform dipertahankan dalam file status yang dibuat secara otomatis. Pengembang Terraform harus berurusan dengan status jauh lebih langsung daripada yang mereka lakukan. CloudFormation Yang penting untuk diingat adalah bahwa status pelacakan sama pentingnya untuk kedua alat.

Secara default, file status Terraform disimpan secara lokal di tingkat atas direktori utama yang menjalankan tumpukan Terraform Anda. Jika Anda menjalankan `terraform apply` perintah dari lingkungan pengembangan lokal Anda, Anda dapat melihat Terraform menghasilkan file terraform.tfstate yang digunakannya untuk mempertahankan status secara real time. Baik atau buruk, ini memberi Anda lebih banyak kendali atas negara bagian di Terraform daripada yang Anda miliki. CloudFormation Meskipun Anda tidak boleh memperbarui file status secara langsung, ada beberapa perintah CLI Terraform yang dapat Anda jalankan yang akan memperbarui status di antara penerapan. Misalnya, terraform import memungkinkan Anda menambahkan sumber daya yang dibuat di luar Terraform ke dalam tumpukan penerapan Anda. Sebaliknya, Anda dapat menghapus sumber daya dari status dengan menjalankan terraform state rm.

Fakta bahwa Terraform perlu menyimpan statusnya di suatu tempat mengarah ke konsep lain yang tidak berlaku untuk CloudFormation: backend. Backend Terraform adalah tempat tumpukan Terraform menyimpan file statusnya setelah penerapan. Ini juga di mana ia mengharapkan untuk menemukan file status ketika penerapan baru dimulai. Saat Anda menjalankan tumpukan secara lokal, seperti dijelaskan di atas, Anda dapat menyimpan salinan status Terraform di direktori lokal tingkat atas. Ini dikenal sebagai backend lokal.

Saat mengembangkan untuk integrasi berkelanjutan dan lingkungan penerapan berkelanjutan (CI/CD), file status lokal umumnya disertakan dalam file.gitignore untuk menjauhkannya dari kontrol versi. Maka tidak ada file status lokal yang ada di dalam pipeline. Agar berfungsi dengan baik, tahap pipeline itu perlu menemukan file status yang benar di suatu tempat.Inilah sebabnya mengapa file

konfigurasi Terraform sering berisi blok backend. Blok backend menunjukkan ke tumpukan Terraform bahwa ia perlu mencari di suatu tempat selain direktori tingkat atasnya sendiri untuk menemukan file status.

Backend Terraform dapat ditemukan hampir di mana saja: bucket Amazon S3, titik akhir API, atau bahkan ruang kerja Terraform jarak jauh. Berikut ini adalah contoh backend Terraform yang disimpan di bucket Amazon S3.

```
terraform {
  backend "s3" {
    bucket = "my-s3-bucket"
    key    = "state-file-folder"
    region = "us-east-1"
  }
}
```

Untuk menghindari penyimpanan informasi sensitif dalam file konfigurasi Terraform, backend juga mendukung konfigurasi sebagian. Pada contoh sebelumnya, kredensional yang diperlukan untuk mengakses bucket tidak ada dalam konfigurasi. Kredensi dapat diperoleh dari variabel lingkungan atau dengan menggunakan cara lain, seperti. AWS Secrets Manager Untuk informasi selengkapnya, lihat Mengamankan data sensitif dengan menggunakan AWS Secrets Manager dan HashiCorp Terraform.

Skenario backend yang umum adalah backend lokal yang digunakan di lingkungan lokal Anda untuk tujuan pengujian. File terraform.tfstate disertakan dalam file.gitignore sehingga tidak didorong ke repositori jarak jauh. Kemudian, setiap lingkungan dalam pipa CI/CD akan mempertahankan backendnya sendiri. Dalam skenario ini, beberapa pengembang mungkin memiliki akses ke status jarak jauh ini, jadi Anda ingin melindungi integritas file status. Jika beberapa penerapan berjalan dan memperbarui status pada saat yang sama, file status dapat menjadi rusak. Untuk alasan ini, dalam situasi dengan backend non-lokal, file status biasanya dikunci selama penerapan.

Memahami sumber data Terraform

Ini sangat umum untuk tumpukan penyebaran untuk mengandalkan data dari sumber daya yang sudah ada sebelumnya. Sebagian besar alat IAC memiliki cara untuk mengimpor sumber daya yang dibuat oleh beberapa proses lain. Sumber daya yang diimpor ini biasanya hanya dibaca (meskipun peran IAM adalah pengecualian penting) dan digunakan untuk mengakses data yang dibutuhkan oleh sumber daya dalam tumpukan. AWS CloudFormation memungkinkan untuk mengimpor sumber daya, tetapi ide ini dapat dijelaskan dengan lebih baik dengan melihat. AWS Cloud Development Kit (AWS CDK)

AWS CDK Ini membantu pengembang menggunakan bahasa pemrograman yang ada untuk menghasilkan CloudFormation template. Hasil akhir dari suatu AWS CDK operasi adalah sumber daya yang diimpor di CloudFormation. Namun sintaks yang digunakan dengan AWS CDK membuat perbandingan yang lebih mudah dengan Terraform. Berikut adalah contoh mengimpor sumber daya dengan menggunakan file. AWS CDK

```
const importedBucket: IBucket = Bucket.fromBucketAttributes(
    scope,
    "imported-bucket",
    {
        bucketName: "My_S3_Bucket"
    }
);
```

Sumber daya yang diimpor biasanya dibuat dengan memanggil metode statis pada kelas yang sama yang Anda gunakan untuk membuat sumber daya baru dari jenis yang sama. Memanggil `new Bucket(...` akan membuat sumber daya baru, dan memanggil `Bucket.fromBucketAttributes(...` impor yang sudah ada. Anda meneruskan subset properti bucket ke dalam fungsi sehingga AWS CDK dapat menemukan bucket yang tepat. Perbedaan lain, bagaimanapun, adalah bahwa membuat bucket baru mengembalikan instance lengkap `Bucket` kelas, dengan semua properti dan metode yang tersedia di dalamnya. Mengimpor sumber daya mengembalikan`IBucket`, yang merupakan tipe yang hanya berisi properti yang `Bucket` harus dimiliki. Meskipun Anda dapat mengimpor sumber daya dari tumpukan eksternal, opsi apa yang dapat Anda lakukan dengannya terbatas.

Di Terraform, tujuan serupa dicapai dengan menggunakan sumber data. Sebagian besar sumber daya Terraform yang ditentukan memiliki sumber data yang menyertainya yang tersedia di

sampingnya. Berikut ini adalah contoh sumber daya bucket Terraform S3 diikuti oleh sumber data yang sesuai.

```
# S3 Bucket resource:
resource "aws_s3_bucket" "My_S3_Bucket" {
  bucket = "My_S3_Bucket"
}

# S3 Bucket data source:
data "aws_s3_bucket" "My_S3_Bucket" {
  bucket = "My_S3_Bucket"
}
```

Satu-satunya perbedaan antara kedua item ini adalah awalan nama. Seperti yang ditunjukkan dalam dokumentasi untuk sumber data, ada lebih sedikit parameter yang tersedia yang dapat Anda berikan ke sumber data daripada sumber daya. Ini karena sumber daya menggunakan parameter tersebut untuk mendeklarasikan semua properti bucket S3 baru, sedangkan sumber data hanya membutuhkan informasi yang cukup untuk mengidentifikasi dan mengimpor data sumber daya yang ada secara unik.

Kesamaan antara sintaks sumber daya Terraform dan sumber data bisa nyaman, tetapi juga bisa bermasalah. Adalah umum bagi pengembang Terraform pemula untuk secara tidak sengaja menggunakan sumber data daripada sumber daya dalam konfigurasi mereka. Sumber data Terraform selalu hanya dibaca. Anda dapat menggunakannya sebagai pengganti sumber daya yang sesuai untuk tindakan baca (seperti memberikan nama ID ke sumber daya lain). Namun, Anda tidak dapat menggunakannya untuk tindakan menulis, yang secara fundamental mengubah beberapa aspek sumber daya yang mendasarinya. Untuk alasan ini, Anda dapat menganggap sumber data Terraform sebagai versi kloning dari sumber daya yang mendasarinya.

Mirip dengan contoh AWS CDK iBucket sebelumnya, sumber data berguna untuk skenario hanya-baca. Jika Anda perlu mendapatkan data dari sumber daya yang ada tetapi tidak perlu mempertahankan sumber daya itu dalam tumpukan Anda, gunakan sumber data. Contoh yang baik dari hal ini adalah ketika Anda membuat instans Amazon EC2 yang menggunakan VPC default akun. Karena VPC itu sudah ada, yang perlu Anda lakukan hanyalah menarik datanya. Contoh kode berikut menunjukkan cara menggunakan data untuk mengidentifikasi VPC target.

```
data "aws_vpc" "default" {
  default = true
}
```

```
resource "aws_instance" "instance1" {
  ami           = "ami-123456"
  instance_type = "t2.micro"
  subnet_id     = data.aws_vpc.default.main_route_table_id
}
```

Memahami variabel Terraform, nilai lokal, dan output

Variabel meningkatkan fleksibilitas kode dengan memungkinkan placeholder dalam blok kode. Variabel dapat mewakili nilai yang berbeda setiap kali kode digunakan kembali. Terraform membedakan antara tipe variabelnya berdasarkan ruang lingkup modularnya. Variabel input adalah nilai eksternal yang dapat disuntikkan ke dalam modul, nilai output adalah nilai internal yang dapat dibagi secara eksternal, dan nilai lokal selalu berada dalam lingkup aslinya.

Variabel

AWS CloudFormation menggunakan parameter untuk mewakili nilai kustom yang dapat diatur dan diatur ulang dari satu penyebaran tumpukan ke yang berikutnya. Demikian pula, Terraform menggunakan variabel input, atau variabel. Variabel dapat dideklarasikan di mana saja dalam file konfigurasi Terraform dan biasanya dideklarasikan dengan tipe data yang diperlukan atau nilai default. Ketiga ekspresi berikut adalah deklarasi variabel Terraform yang valid.

```
variable "thing_i_made_up" {
  type = string
}

variable "random_number" {
  default = 5
}

variable "dogs" {
  type = list(object({
    name  = string
    breed = string
  }))

  default = [
    {
      name  = "Sparky",
      breed = "poodle"
    }
  ]
}
```

Untuk mengakses breed Sparky dalam konfigurasi, Anda akan menggunakan variabel`var.dogs[0].breed`. Jika variabel tidak memiliki default dan tidak diklasifikasikan sebagai nullable, maka nilai variabel harus ditetapkan untuk setiap penerapan. Jika tidak, itu opsional untuk menetapkan nilai baru untuk variabel. Dalam modul root, Anda dapat mengatur nilai variabel saat ini pada baris perintah, sebagai variabel lingkungan, atau dalam file terraform.tfvars. Contoh berikut menunjukkan cara memasukkan nilai variabel dalam file terraform.tfvars, yang disimpan di direktori tingkat atas modul.

```
# terraform.tfvars
dogs = [
   {
      name  = "Sparky",
      breed = "poodle"
   },
   {
      name  = "Fluffy",
      breed = "chihuahua"
   }
]

random_number = 7

thing_i_made_up = "Kabibble"
```

Nilai untuk `dogs` dalam contoh file terraform.tfvars ini akan mengganti nilai default dalam deklarasi variabel. Jika Anda mendeklarasikan variabel dalam modul anak, Anda dapat mengatur nilai variabel langsung dalam blok deklarasi modul, seperti yang ditunjukkan pada contoh berikut.

```
module "my_custom_module" {
  source        = "modulesource/custom"
  version       = "0.0.1"
  random_number = 8
}
```

Beberapa argumen lain yang dapat Anda gunakan saat mendeklarasikan variabel meliputi:

- `sensitive`— Menyetel ini untuk `true` mencegah nilai variabel terekspos dalam output proses Terraform.
- `nullable`— Mengatur ini untuk `true` memungkinkan variabel tidak memiliki nilai. Ini nyaman untuk variabel di mana default tidak diatur.

- `description`— Tambahkan deskripsi variabel ke metadata untuk tumpukan.
- `validation`— Tetapkan aturan validasi untuk variabel.

Salah satu aspek yang paling nyaman dari variabel Terraform adalah kemampuan untuk menambahkan satu atau lebih objek validasi dalam deklarasi variabel. Anda dapat menggunakan objek validasi untuk menambahkan kondisi yang harus dilewati variabel atau penerapan gagal. Anda juga dapat mengatur pesan kesalahan kustom untuk ditampilkan setiap kali kondisi dilanggar.

Misalnya, Anda menyiapkan file konfigurasi Terraform yang akan dijalankan oleh anggota tim Anda. Sebelum menerapkan tumpukan, anggota tim perlu membuat file terraform.tfvars untuk menetapkan nilai konfigurasi penting. Untuk mengingatkan mereka, Anda bisa melakukan sesuatu seperti berikut ini.

```
variable "important_config_setting" {
  type = string

  validation {
    condition     = length(var.important_config_setting) > 0
    error_message = "Don't forget to create the terraform.tfvars file!"
  }

  validation {
    condition     = substr(var.important_config_setting, 0, 7) == "prefix-"
    error_message = "Remember that the value always needs to start with 'prefix-'"
  }
}
```

Seperti yang ditunjukkan dalam contoh ini, Anda dapat mengatur beberapa kondisi di dalam variabel tunggal. Terraform hanya menampilkan pesan kesalahan untuk kondisi gagal. Dengan cara ini, Anda dapat menegakkan semua jenis aturan pada nilai variabel. Jika nilai variabel menyebabkan kegagalan pipeline, Anda akan tahu persis mengapa.

Nilai-nilai lokal

Jika ada nilai dalam modul yang Anda ingin alias, gunakan `locals` kata kunci daripada mendeklarasikan variabel default yang tidak akan pernah diperbarui. Seperti namanya, `locals` blok berisi istilah yang dicakup secara internal ke modul tertentu. Jika Anda ingin mengubah nilai string, seperti dengan menambahkan awalan ke nilai variabel untuk digunakan dalam nama sumber

daya, menggunakan nilai lokal mungkin merupakan solusi yang baik. Satu `locals` blok dapat mendeklarasikan semua nilai lokal untuk modul Anda, seperti yang ditunjukkan pada contoh berikut.

```
locals {
  moduleName    = "My Module"
  localConfigId = concat("prefix-", var.important_config_setting)
}
```

Ingatlah bahwa ketika Anda mengakses nilai, `locals` kata kunci menjadi tunggal, seperti. `local.LocalConfigId`

Nilai keluaran

Jika variabel input Terraform seperti CloudFormation parameter, maka Anda dapat mengatakan bahwa nilai keluaran Terraform seperti output. CloudFormation Keduanya digunakan untuk mengekspos nilai dari dalam tumpukan penyebaran. Namun, karena modul Terraform lebih mendarah daging ke dalam struktur alat, nilai keluaran Terraform juga digunakan untuk mengekspos nilai dalam modul ke modul induk atau modul anak lainnya, bahkan jika modul tersebut semuanya berada dalam tumpukan penerapan yang sama. Jika Anda sedang membangun dua modul kustom dan modul pertama perlu mengakses nilai ID dari modul kedua, maka Anda harus menambahkan `output` blok berikut ke modul kedua.

```
output "module_id" {
  value = local.module_id
}
Then in the first module you could use it like this:
module "first_module" {
  source = "path/to/first/module"
}

resource "example_resource" "example_resource_name" {
  module_id = module.first_module.module_id
}
```

Karena nilai keluaran Terraform dapat digunakan dalam tumpukan yang sama, Anda juga dapat menggunakan `sensitive` atribut dalam `output` blok untuk menekan nilai agar tidak ditampilkan dalam output tumpukan. Selain itu, `output` blok dapat menggunakan `precondition` blok dengan cara yang sama seperti variabel menggunakan `validation` blok: untuk memastikan variabel

mengikuti seperangkat aturan tertentu. Ini membantu memastikan bahwa semua nilai dalam modul ada seperti yang diharapkan sebelum melanjutkan dengan penerapan.

```
output "important_config_setting" {
  value = var.important_config_setting

  precondition {
    condition     = length(var.important_config_setting) > 0
    error_message = "You forgot to create the terraform.tfvars file again."
  }
}
```

Memahami fungsi, ekspresi, dan meta-argumen Terraform

Salah satu kritik terhadap alat IAC yang menggunakan file konfigurasi deklaratif daripada bahasa pemrograman umum adalah bahwa mereka membuatnya lebih sulit untuk mengimplementasikan logika programatik khusus. Dalam konfigurasi Terraform, masalah ini diatasi dengan menggunakan fungsi, ekspresi, dan meta-argumen.

Fungsi

Salah satu keuntungan besar menggunakan kode untuk menyediakan infrastruktur Anda adalah kemampuan untuk menyimpan alur kerja umum dan menggunakannya kembali dan lagi, sering kali melewati argumen yang berbeda setiap kali. Fungsi Terraform mirip dengan fungsi AWS CloudFormation intrinsik, meskipun sintaksnya lebih mirip dengan bagaimana fungsi dipanggil dalam bahasa terprogram. Anda mungkin telah memperhatikan beberapa fungsi Terraform, seperti seperti substr, concat, length, dan base64decode, dalam contoh dalam panduan ini. CloudFormation Seperti fungsi intrinsik, Terraform memiliki serangkaian fungsi bawaan yang tersedia untuk digunakan dalam konfigurasi Anda. Misalnya, jika atribut sumber daya tertentu mengambil objek JSON yang sangat besar yang tidak efisien untuk ditempelkan langsung ke file, Anda dapat meletakkan objek tersebut dalam file.json dan menggunakan fungsi Terraform untuk mengaksesnya. Dalam contoh berikut, `file` fungsi mengembalikan isi file dalam bentuk string, dan kemudian `jsondecode` fungsi mengubahnya menjadi tipe objek.

```
resource "example_resource" "example_resource_name" {
  json_object = jsondecode(file("/path/to/file.json"))
}
```

Ekspresi

Terraform juga memungkinkan ekspresi bersyarat, yang mirip dengan CloudFormation `condition` fungsi kecuali bahwa mereka menggunakan sintaks operator ternary yang lebih tradisional. Dalam contoh berikut, kedua ekspresi mengembalikan hasil yang sama persis. Contoh kedua adalah apa yang disebut Terraform sebagai ekspresi percikan. Tanda bintang menyebabkan Terraform mengulang daftar dan membuat daftar baru hanya dengan menggunakan `id` properti setiap item.

```
resource "example_resource" "example_resource_name" {
  boolean_value  = var.value ? true : false
```

```
  numeric_value  = var.value > 0 ? 1 : 0
  string_value   = var.value == "change_me" ? "New value" : var.value
  string_value_2 = var.value != "change_me" ? var.value : "New value"
}
There are two ways to express for loops in a Terraform configuration:
resource "example_resource" "example_resource_name" {
  list_value   = [for object in var.ids : object.id]
  list_value_2 = var.ids[*].id
}
```

Meta-argumen

Dalam contoh kode sebelumnya, `list_value` dan `list_value_2` disebut sebagai argumen. Anda mungkin sudah akrab dengan beberapa meta-argumen ini. Terraform juga memiliki beberapa meta-argumen, yang bertindak seperti argumen tetapi dengan beberapa fungsionalitas tambahan:

- Meta-argument depends_on sangat mirip dengan atribut. CloudFormation DependsOn
- Meta-argument penyedia memungkinkan Anda menggunakan beberapa konfigurasi penyedia sekaligus.
- Meta-argumen siklus hidup memungkinkan Anda untuk menyesuaikan setelan sumber daya, mirip dengan kebijakan penghapusan dan penghapusan di. CloudFormation

Meta-argumen lain memungkinkan fungsi dan ekspresi fungsionalitas ditambahkan langsung ke sumber daya. Misalnya, hitungan meta-argumen adalah mekanisme yang berguna untuk membuat beberapa sumber daya serupa pada saat yang bersamaan. Contoh berikut menunjukkan cara membuat dua cluster Amazon Elastic Container Service (Amazon EKS) tanpa menggunakan `count` meta-argumen.

```
resource "aws_eks_cluster" "example_0" {
  name     = "example_0"
  role_arn = aws_iam_role.cluster_role.arn
  vpc_config {
    endpoint_private_access = true
    endpoint_public_access  = true
    subnet_ids              = var.subnet_ids[0]
  }
}

resource "aws_eks_cluster" "example_1" {
```

```
  name     = "example_1"
  role_arn = aws_iam_role.cluster_role.arn
  vpc_config {
    endpoint_private_access = true
    endpoint_public_access  = true
    subnet_ids              = var.subnet_ids[1]
  }
}
```

Contoh berikut menunjukkan cara menggunakan `count` meta-argumen untuk membuat dua cluster Amazon EKS.

```
resource "aws_eks_cluster" "clusters" {
  count    = 2
  name     = "cluster_${count.index}"
  role_arn = aws_iam_role.cluster_role.arn
  vpc_config {
    endpoint_private_access = true
    endpoint_public_access  = true
    subnet_ids              = var.subnet_ids[count.index]
  }
}
```

Untuk memberikan masing-masing nama unit, Anda dapat mengakses indeks daftar dalam blok sumber daya di`count.index`. Tetapi bagaimana jika Anda ingin membuat beberapa sumber daya serupa yang sedikit lebih kompleks? Di situlah meta-argumen for_each masuk. `for_each`Meta-argumen sangat mirip dengan`count`, kecuali bahwa Anda meneruskan daftar atau objek bukan angka. Terraform membuat sumber daya baru untuk setiap anggota daftar atau objek. Ini mirip dengan jika Anda mengatur`count = length(list)`, kecuali Anda dapat mengakses isi daftar daripada indeks loop.

Ini berfungsi untuk daftar item atau objek tunggal. Contoh berikut akan membuat dua sumber daya yang memiliki `id-0` dan `id-1` sebagai ID mereka.

```
variable "ids" {
  default = [
    { id = "id-0" },
    { id = "id-1" },
  ]
}
```

```
resource "example_resource" "example_resource_name" {
  # If your list fails, you might have to call "toset" on it to convert it to a set
  for_each = toset(var.ids)
  id       = each.value
}
```

Contoh berikut akan menciptakan dua sumber daya juga, satu untuk Sparky, pudel, dan satu untuk Fluffy, chihuahua.

```
variable "dogs" {
  default = {
    poodle    = "Sparky"
    chihuahua = "Fluffy"
  }
}

resource "example_resource" "example_resource_name" {
  for_each = var.dogs
  breed    = each.key
  name     = each.value
}
```

Sama seperti Anda dapat mengakses indeks loop dalam hitungan dengan menggunakan count.index, Anda dapat mengakses kunci dan nilai setiap item dalam loop for_each dengan menggunakan setiap objek. Karena for_each mengulangi daftar dan objek, setiap kunci dan nilai bisa sedikit membingungkan untuk dilacak. Tabel berikut menunjukkan berbagai cara yang dapat Anda gunakan meta-argumen for_each dan bagaimana Anda dapat mereferensikan nilai pada setiap iterasi.

Contoh	**for_each**jenis	Iterasi pertama	Iterasi kedua

A	`[“poodle”, “chihuahua”]`	`each.key = "poodle"`
		
		`each.value = null`
			`each.key = "chihuahua"`
			
			`each.value = null`

B	`[`	`each.key = {`	`each.key = {`

Contoh	**for_each**jenis	Iterasi pertama	Iterasi kedua

	{
	
	type =
	"poodle",
	
	name = "Sparky"
	
	},
	
	{
	
	type = "chihuahu
	a",
	
	name = "Fluffy"
	
	}
	
	]
		type = “poodle”,
		
		name = “Sparky”
		
		}
		
		each.value =
		null
			type = “chihuahu
			a”,
			
			name = “Fluffy”
			
			}
			
			each.value =
			null

C	{
	
	poodle =
	“Sparky”,
	
	chihuahua =
	“Fluffy”
	
	}
		each.key =
		“poodle”
		
		each.value =
		“Sparky”
			each.key =
			“chihuahua”
			
			each.value =
			“Fluffy”

Contoh	**for_each**jenis	Iterasi pertama	Iterasi kedua

D	`{`
	
	`dogs = {`
	
	`poodle =`
	` “Sparky”,`
	
	`chihuahua =`
	` “Fluffy”`
	
	`},`
	
	`cats = {`
	
	`persian =`
	` “Felix”,`
	
	`burmese =`
	` “Morris”`
	
	`}`
	
	`}`
		`each.key = “dogs”`
		
		`each.value = {`
		
		`poodle =`
		` “Sparky”,`
		
		`chihuahua =`
		` “Fluffy”`
		
		`}`
			`each.key = “cats”`
			
			`each.value = {`
			
			`persian =`
			` “Felix”,`
			
			`burmese =`
			` “Morris”`
			
			`}`

Contoh	**for_each**jenis	Iterasi pertama	Iterasi kedua

E	{
	
	dogs = [
	
	{
	
	type = “poodle”,
	
	name = “Sparky”
	
	},
	
	{
	
	type = “chihuahua”,
	
	name = “Fluffy”
	
	}
	
	],
	
	cats = [
	
	{
	
	type = “persian”,
	
	name = “Felix”
	
	},
	
	{
	
	type = “burmese”,
		each.key = “dogs”
		
		each.value = [
		
		{
		
		type = “poodle”,
		
		name = “Sparky”
		
		},
		
		{
		
		type = “chihuahua”,
		
		name = “Fluffy”
		
		}
		
		]
			each.key = “cats”
			
			each.value = [
			
			{
			
			type = “persian”,
			
			name = “Felix”
			
			},
			
			{
			
			type = “burmese”,
			
			name = “Morris”
			
			}
			
			]

Contoh	**for_each**jenis	Iterasi pertama	Iterasi kedua
	name = “Morris” }] }		

Jadi jika `var.animals` sama dengan baris E, maka Anda bisa membuat satu sumber daya per hewan dengan menggunakan kode berikut.

```
resource "example_resource" "example_resource_name" {
  for_each = var.animals
  type     = each.key
  breeds   = each.value[*].type
  names    = each.value[*].name
}
```

Atau, Anda dapat membuat dua sumber daya per hewan dengan menggunakan kode berikut.

```
resource "example_resource" "example_resource_name" {
  for_each = var.animals.dogs
  type     = "dogs"
  breeds   = each.value.type
  names    = each.value.name
}

resource "example_resource" "example_resource_name" {
  for_each = var.animals.cats
  type     = "cats"
  breeds   = each.value.type
  names    = each.value.name
}
```

Pertanyaan yang Sering Diajukan

Kapan saya harus menggunakan Terraform alih-alih? CloudFormation

Secara umum, jika beban kerja Anda didasarkan pada AWS, AWS CloudFormation berikan tingkat dukungan asli yang tidak dapat ditandingi oleh Terraform. Namun, jika beban kerja Anda mencakup beberapa proses pihak ketiga atau tersebar di antara beberapa penyedia cloud, Terraform adalah alat yang mungkin ingin Anda pertimbangkan.

Kapan saya harus menggunakan AWS CDK alih-alih CloudFormation?

Ketika Anda menggunakan AWS Cloud Development Kit (AWS CDK), Anda juga menggunakan CloudFormation. AWS CDK Ini memungkinkan Anda untuk menggunakan bahasa pemrograman umum untuk menghasilkan CloudFormation template. Jika Anda berpengalaman dalam salah satu bahasa pemrograman yang AWS CDK didukung, AWS CDK dapat mengurangi waktu yang diperlukan untuk menghasilkan CloudFormation template.

Apakah ada alat seperti AWS CDK yang menghasilkan konfigurasi Terraform?

Dibandingkan dengan AWS CDK, CDK untuk Terraform (CDKTF) menggunakan pustaka konstruksi yang sama untuk menyediakan sumber daya dan mesin jsii yang sama untuk mendukung berbagai bahasa pemrograman. Anda dapat menggunakannya untuk menghasilkan konfigurasi Terraform dengan cara yang sama seperti yang menghasilkan templat. AWS CDK CloudFormation

Bagaimana cara mempelajari lebih lanjut tentang Terraform?

Untuk informasi lebih lanjut tentang konsep Terraform tingkat lanjut, lihat dokumentasi Terraform. Ini juga menjelaskan komponen dari semua penyedia utama dan modul open source.

Sumber daya terkait

AWS dokumentasi

- Dokumentasi AWS CDK
- Dokumentasi AWS CloudFormation
- Terraform: Melampaui Dasar-Dasar dengan AWS (AWS posting blog)

Sumber daya lainnya

- CDK untuk dokumentasi Terraform
- Dokumentasi Terraform

Lampiran: Contoh akses atribut Terraform

Sumber Daya

```
resource "aws_s3_bucket" "myS3Bucket" {
     bucket = "my-s3-bucket"
}

bucketName = aws_s3_bucket.myS3Bucket.bucket
```

Sumber data

```
data "aws_s3_bucket" "myS3Bucket" {
     bucket = "my-s3-bucket"
}

bucketName = data.aws_s3_bucket.myS3Bucket.bucket
```

Modul

```
module "eks" {
    source = "terraform-aws-modules/eks/aws"
    version = "20.2.1"
}

vpc_id = module.eks.vpc_id
```

Variabel

```
variable "my_variable" = {
   default = "dog"
}

animalType = var.my_variable
```

Lokal:

```
locals {
  type = "dog"
}

animalType = local.type
```

Riwayat dokumen

Tabel berikut menjelaskan perubahan signifikan pada panduan ini. Jika Anda ingin diberi tahu tentang pembaruan masa depan, Anda dapat berlangganan umpan RSS.

Perubahan	Deskripsi	Tanggal
Publikasi awal	—	Maret 29, 2024

AWS Glosarium Panduan Preskriptif

Berikut ini adalah istilah yang umum digunakan dalam strategi, panduan, dan pola yang disediakan oleh Panduan AWS Preskriptif. Untuk menyarankan entri, silakan gunakan tautan Berikan umpan balik di akhir glosarium.

Nomor

7 Rs

Tujuh strategi migrasi umum untuk memindahkan aplikasi ke cloud. Strategi ini dibangun di atas 5 Rs yang diidentifikasi Gartner pada tahun 2011 dan terdiri dari yang berikut:

- Refactor/Re-Architect — Memindahkan aplikasi dan memodifikasi arsitekturnya dengan memanfaatkan sepenuhnya fitur cloud-native untuk meningkatkan kelincahan, kinerja, dan skalabilitas. Ini biasanya melibatkan porting sistem operasi dan database. Contoh: Migrasikan database Oracle lokal Anda ke Amazon Aurora PostgreSQL Compatible Edition.
- Replatform (angkat dan bentuk ulang) — Pindahkan aplikasi ke cloud, dan perkenalkan beberapa tingkat pengoptimalan untuk memanfaatkan kemampuan cloud. Contoh: Migrasikan database Oracle lokal Anda ke Amazon Relational Database Service (Amazon RDS) untuk Oracle di. AWS Cloud
- Pembelian kembali (drop and shop) - Beralih ke produk yang berbeda, biasanya dengan beralih dari lisensi tradisional ke model SaaS. Contoh: Migrasikan sistem manajemen hubungan pelanggan (CRM) Anda ke Salesforce.com.
- Rehost (lift dan shift) — Pindahkan aplikasi ke cloud tanpa membuat perubahan apa pun untuk memanfaatkan kemampuan cloud. Contoh: Migrasikan database Oracle lokal Anda ke Oracle pada instans EC2 di. AWS Cloud
- Relokasi (hypervisor-level lift and shift) — Pindahkan infrastruktur ke cloud tanpa membeli perangkat keras baru, menulis ulang aplikasi, atau memodifikasi operasi yang ada. Anda memigrasikan server dari platform lokal ke layanan cloud untuk platform yang sama. Contoh: Migrasikan Microsoft Hyper-V aplikasi ke AWS.
- Pertahankan (kunjungi kembali) - Simpan aplikasi di lingkungan sumber Anda. Ini mungkin termasuk aplikasi yang memerlukan refactoring besar, dan Anda ingin menunda pekerjaan itu sampai nanti, dan aplikasi lama yang ingin Anda pertahankan, karena tidak ada pembenaran bisnis untuk memigrasikannya.

- Pensiun — Menonaktifkan atau menghapus aplikasi yang tidak lagi diperlukan di lingkungan sumber Anda.

A

ABAC

Lihat kontrol akses berbasis atribut.

layanan abstrak

Lihat layanan terkelola.

ASAM

Lihat atomisitas, konsistensi, isolasi, daya tahan.

migrasi aktif-aktif

Metode migrasi database di mana database sumber dan target tetap sinkron (dengan menggunakan alat replikasi dua arah atau operasi penulisan ganda), dan kedua database menangani transaksi dari menghubungkan aplikasi selama migrasi. Metode ini mendukung migrasi dalam batch kecil yang terkontrol alih-alih memerlukan pemotongan satu kali. Ini lebih fleksibel tetapi membutuhkan lebih banyak pekerjaan daripada migrasi aktif-pasif.

migrasi aktif-pasif

Metode migrasi database di mana database sumber dan target disimpan dalam sinkron, tetapi hanya database sumber yang menangani transaksi dari menghubungkan aplikasi sementara data direplikasi ke database target. Basis data target tidak menerima transaksi apa pun selama migrasi.

fungsi agregat

Fungsi SQL yang beroperasi pada sekelompok baris dan menghitung nilai pengembalian tunggal untuk grup. Contoh fungsi agregat meliputi `SUM` dan`MAX`.

AI

Lihat kecerdasan buatan.

AIOps

Lihat operasi kecerdasan buatan.

anonimisasi

Proses menghapus informasi pribadi secara permanen dalam kumpulan data. Anonimisasi dapat membantu melindungi privasi pribadi. Data anonim tidak lagi dianggap sebagai data pribadi.

anti-pola

Solusi yang sering digunakan untuk masalah berulang di mana solusinya kontra-produktif, tidak efektif, atau kurang efektif daripada alternatif.

kontrol aplikasi

Pendekatan keamanan yang memungkinkan penggunaan hanya aplikasi yang disetujui untuk membantu melindungi sistem dari malware.

portofolio aplikasi

Kumpulan informasi rinci tentang setiap aplikasi yang digunakan oleh organisasi, termasuk biaya untuk membangun dan memelihara aplikasi, dan nilai bisnisnya. Informasi ini adalah kunci untuk penemuan portofolio dan proses analisis dan membantu mengidentifikasi dan memprioritaskan aplikasi yang akan dimigrasi, dimodernisasi, dan dioptimalkan.

kecerdasan buatan (AI)

Bidang ilmu komputer yang didedikasikan untuk menggunakan teknologi komputasi untuk melakukan fungsi kognitif yang biasanya terkait dengan manusia, seperti belajar, memecahkan masalah, dan mengenali pola. Untuk informasi lebih lanjut, lihat Apa itu Kecerdasan Buatan?

operasi kecerdasan buatan (AIOps)

Proses menggunakan teknik pembelajaran mesin untuk memecahkan masalah operasional, mengurangi insiden operasional dan intervensi manusia, dan meningkatkan kualitas layanan. Untuk informasi selengkapnya tentang cara AIOps digunakan dalam strategi AWS migrasi, lihat panduan integrasi operasi.

enkripsi asimetris

Algoritma enkripsi yang menggunakan sepasang kunci, kunci publik untuk enkripsi dan kunci pribadi untuk dekripsi. Anda dapat berbagi kunci publik karena tidak digunakan untuk dekripsi, tetapi akses ke kunci pribadi harus sangat dibatasi.

atomisitas, konsistensi, isolasi, daya tahan (ACID)

Satu set properti perangkat lunak yang menjamin validitas data dan keandalan operasional database, bahkan dalam kasus kesalahan, kegagalan daya, atau masalah lainnya.

kontrol akses berbasis atribut (ABAC)

Praktik membuat izin berbutir halus berdasarkan atribut pengguna, seperti departemen, peran pekerjaan, dan nama tim. Untuk informasi selengkapnya, lihat ABAC untuk AWS dokumentasi AWS Identity and Access Management (IAM).

sumber data otoritatif

Lokasi di mana Anda menyimpan versi utama data, yang dianggap sebagai sumber informasi yang paling dapat diandalkan. Anda dapat menyalin data dari sumber data otoritatif ke lokasi lain untuk tujuan memproses atau memodifikasi data, seperti menganonimkan, menyunting, atau membuat nama samaran.

Zona Ketersediaan

Lokasi berbeda di dalam Wilayah AWS yang terisolasi dari kegagalan di Availability Zone lainnya dan menyediakan konektivitas jaringan latensi rendah yang murah ke Availability Zone lainnya di Wilayah yang sama.

AWS Kerangka Adopsi Cloud (AWS CAF)

Kerangka pedoman dan praktik terbaik AWS untuk membantu organisasi mengembangkan rencana yang efisien dan efektif untuk bergerak dengan sukses ke cloud. AWS CAF mengatur panduan ke dalam enam area fokus yang disebut perspektif: bisnis, orang, tata kelola, platform, keamanan, dan operasi. Perspektif bisnis, orang, dan tata kelola fokus pada keterampilan dan proses bisnis; perspektif platform, keamanan, dan operasi fokus pada keterampilan dan proses teknis. Misalnya, perspektif masyarakat menargetkan pemangku kepentingan yang menangani sumber daya manusia (SDM), fungsi kepegawaian, dan manajemen orang. Untuk perspektif ini, AWS CAF memberikan panduan untuk pengembangan, pelatihan, dan komunikasi orang untuk membantu mempersiapkan organisasi untuk adopsi cloud yang sukses. Untuk informasi lebih lanjut, lihat situs web AWS CAF dan whitepaper AWS CAF.

AWS Kerangka Kualifikasi Beban Kerja (AWS WQF)

Alat yang mengevaluasi beban kerja migrasi database, merekomendasikan strategi migrasi, dan memberikan perkiraan kerja. AWS WQF disertakan dengan AWS Schema Conversion Tool ()AWS SCT. Ini menganalisis skema database dan objek kode, kode aplikasi, dependensi, dan karakteristik kinerja, dan memberikan laporan penilaian.

B

bot buruk

Bot yang dimaksudkan untuk mengganggu atau menyebabkan kerugian bagi individu atau organisasi.

BCP

Lihat perencanaan kontinuitas bisnis.

grafik perilaku

Pandangan interaktif yang terpadu tentang perilaku dan interaksi sumber daya dari waktu ke waktu. Anda dapat menggunakan grafik perilaku dengan Amazon Detective untuk memeriksa upaya logon yang gagal, panggilan API yang mencurigakan, dan tindakan serupa. Untuk informasi selengkapnya, lihat Data dalam grafik perilaku di dokumentasi Detektif.

sistem big-endian

Sistem yang menyimpan byte paling signifikan terlebih dahulu. Lihat juga endianness.

klasifikasi biner

Sebuah proses yang memprediksi hasil biner (salah satu dari dua kelas yang mungkin). Misalnya, model ML Anda mungkin perlu memprediksi masalah seperti “Apakah email ini spam atau bukan spam?” atau “Apakah produk ini buku atau mobil?”

filter mekar

Struktur data probabilistik dan efisien memori yang digunakan untuk menguji apakah suatu elemen adalah anggota dari suatu himpunan.

deployment biru/hijau

Strategi penyebaran tempat Anda membuat dua lingkungan yang terpisah namun identik. Anda menjalankan versi aplikasi saat ini di satu lingkungan (biru) dan versi aplikasi baru di lingkungan lain (hijau). Strategi ini membantu Anda dengan cepat memutar kembali dengan dampak minimal.

bot

Aplikasi perangkat lunak yang menjalankan tugas otomatis melalui internet dan mensimulasikan aktivitas atau interaksi manusia. Beberapa bot berguna atau bermanfaat, seperti perayap web yang mengindeks informasi di internet. Beberapa bot lain, yang dikenal sebagai bot buruk, dimaksudkan untuk mengganggu atau membahayakan individu atau organisasi.

botnet

Jaringan bot yang terinfeksi oleh malware dan berada di bawah kendali satu pihak, yang dikenal sebagai bot herder atau operator bot. Botnet adalah mekanisme paling terkenal untuk skala bot dan dampaknya.

cabang

Area berisi repositori kode. Cabang pertama yang dibuat dalam repositori adalah cabang utama. Anda dapat membuat cabang baru dari cabang yang ada, dan Anda kemudian dapat mengembangkan fitur atau memperbaiki bug di cabang baru. Cabang yang Anda buat untuk membangun fitur biasanya disebut sebagai cabang fitur. Saat fitur siap dirilis, Anda menggabungkan cabang fitur kembali ke cabang utama. Untuk informasi selengkapnya, lihat Tentang cabang (GitHub dokumentasi).

akses break-glass

Dalam keadaan luar biasa dan melalui proses yang disetujui, cara cepat bagi pengguna untuk mendapatkan akses ke Akun AWS yang biasanya tidak memiliki izin untuk mengaksesnya. Untuk informasi lebih lanjut, lihat indikator Implementasikan prosedur break-glass dalam panduan Well-Architected AWS .

strategi brownfield

Infrastruktur yang ada di lingkungan Anda. Saat mengadopsi strategi brownfield untuk arsitektur sistem, Anda merancang arsitektur di sekitar kendala sistem dan infrastruktur saat ini. Jika Anda memperluas infrastruktur yang ada, Anda dapat memadukan strategi brownfield dan greenfield.

cache penyangga

Area memori tempat data yang paling sering diakses disimpan.

kemampuan bisnis

Apa yang dilakukan bisnis untuk menghasilkan nilai (misalnya, penjualan, layanan pelanggan, atau pemasaran). Arsitektur layanan mikro dan keputusan pengembangan dapat didorong oleh kemampuan bisnis. Untuk informasi selengkapnya, lihat bagian Terorganisir di sekitar kemampuan bisnis dari Menjalankan layanan mikro kontainer di whitepaper. AWS

perencanaan kelangsungan bisnis (BCP)

Rencana yang membahas dampak potensial dari peristiwa yang mengganggu, seperti migrasi skala besar, pada operasi dan memungkinkan bisnis untuk melanjutkan operasi dengan cepat.

C

KAFE

Lihat Kerangka Adopsi AWS Cloud.

penyebaran kenari

Rilis versi yang lambat dan bertahap untuk pengguna akhir. Ketika Anda yakin, Anda menyebarkan versi baru dan mengganti versi saat ini secara keseluruhan.

CCoE

Lihat Cloud Center of Excellence.

CDC

Lihat mengubah pengambilan data.

ubah pengambilan data (CDC)

Proses melacak perubahan ke sumber data, seperti tabel database, dan merekam metadata tentang perubahan tersebut. Anda dapat menggunakan CDC untuk berbagai tujuan, seperti mengaudit atau mereplikasi perubahan dalam sistem target untuk mempertahankan sinkronisasi.

rekayasa kekacauan

Sengaja memperkenalkan kegagalan atau peristiwa yang mengganggu untuk menguji ketahanan sistem. Anda dapat menggunakan AWS Fault Injection Service (AWS FIS) untuk melakukan eksperimen yang menekankan AWS beban kerja Anda dan mengevaluasi responsnya.

CI/CD

Lihat integrasi berkelanjutan dan pengiriman berkelanjutan.

klasifikasi

Proses kategorisasi yang membantu menghasilkan prediksi. Model ML untuk masalah klasifikasi memprediksi nilai diskrit. Nilai diskrit selalu berbeda satu sama lain. Misalnya, model mungkin perlu mengevaluasi apakah ada mobil dalam gambar atau tidak.

Enkripsi sisi klien

Enkripsi data secara lokal, sebelum target Layanan AWS menerimanya.

Cloud Center of Excellence (CCoE)

Tim multi-disiplin yang mendorong upaya adopsi cloud di seluruh organisasi, termasuk mengembangkan praktik terbaik cloud, memobilisasi sumber daya, menetapkan jadwal migrasi, dan memimpin organisasi melalui transformasi skala besar. Untuk informasi selengkapnya, lihat posting CCoE di Blog Strategi AWS Cloud Perusahaan.

komputasi cloud

Teknologi cloud yang biasanya digunakan untuk penyimpanan data jarak jauh dan manajemen perangkat IoT. Cloud computing umumnya terhubung ke teknologi edge computing.

model operasi cloud

Dalam organisasi TI, model operasi yang digunakan untuk membangun, mematangkan, dan mengoptimalkan satu atau lebih lingkungan cloud. Untuk informasi selengkapnya, lihat Membangun Model Operasi Cloud Anda.

tahap adopsi cloud

Empat fase yang biasanya dilalui organisasi ketika mereka bermigrasi ke AWS Cloud:

- Proyek — Menjalankan beberapa proyek terkait cloud untuk bukti konsep dan tujuan pembelajaran
- Foundation — Melakukan investasi dasar untuk meningkatkan adopsi cloud Anda (misalnya, membuat landing zone, mendefinisikan CCoE, membuat model operasi)
- Migrasi — Migrasi aplikasi individual
- Re-invention — Mengoptimalkan produk dan layanan, dan berinovasi di cloud

Tahapan ini didefinisikan oleh Stephen Orban dalam posting blog The Journey Toward Cloud-First & the Stages of Adoption di blog Strategi Perusahaan. AWS Cloud Untuk informasi tentang bagaimana kaitannya dengan strategi AWS migrasi, lihat panduan kesiapan migrasi.

CMDB

Lihat database manajemen konfigurasi.

repositori kode

Lokasi di mana kode sumber dan aset lainnya, seperti dokumentasi, sampel, dan skrip, disimpan dan diperbarui melalui proses kontrol versi. Repositori cloud umum termasuk GitHub atau. AWS CodeCommit Setiap versi kode disebut cabang. Dalam struktur layanan mikro, setiap repositori

dikhususkan untuk satu bagian fungsionalitas. Pipa CI/CD tunggal dapat menggunakan beberapa repositori.

cache dingin

Cache buffer yang kosong, tidak terisi dengan baik, atau berisi data basi atau tidak relevan. Ini mempengaruhi kinerja karena instance database harus membaca dari memori utama atau disk, yang lebih lambat daripada membaca dari cache buffer.

data dingin

Data yang jarang diakses dan biasanya historis. Saat menanyakan jenis data ini, kueri lambat biasanya dapat diterima. Memindahkan data ini ke tingkat penyimpanan atau kelas yang berkinerja lebih rendah dan lebih murah dapat mengurangi biaya.

visi komputer (CV)

Bidang AI yang menggunakan pembelajaran mesin untuk menganalisis dan mengekstrak informasi dari format visual seperti gambar dan video digital. Misalnya, AWS Panorama menawarkan perangkat yang menambahkan CV ke jaringan kamera lokal, dan Amazon SageMaker menyediakan algoritme pemrosesan gambar untuk CV.

konfigurasi drift

Untuk beban kerja, konfigurasi berubah dari status yang diharapkan. Ini dapat menyebabkan beban kerja menjadi tidak patuh, dan biasanya bertahap dan tidak disengaja.

database manajemen konfigurasi (CMDB)

Repositori yang menyimpan dan mengelola informasi tentang database dan lingkungan TI, termasuk komponen perangkat keras dan perangkat lunak dan konfigurasinya. Anda biasanya menggunakan data dari CMDB dalam penemuan portofolio dan tahap analisis migrasi.

paket kesesuaian

Kumpulan AWS Config aturan dan tindakan remediasi yang dapat Anda kumpulkan untuk menyesuaikan kepatuhan dan pemeriksaan keamanan Anda. Anda dapat menerapkan paket kesesuaian sebagai entitas tunggal di Akun AWS dan Wilayah, atau di seluruh organisasi, dengan menggunakan templat YAMM. Untuk informasi selengkapnya, lihat Paket kesesuaian dalam dokumentasi. AWS Config

integrasi berkelanjutan dan pengiriman berkelanjutan (CI/CD)

Proses mengotomatiskan sumber, membangun, menguji, pementasan, dan tahap produksi dari proses rilis perangkat lunak. CI/CD umumnya digambarkan sebagai pipa. CI/CD dapat membantu

Anda mengotomatiskan proses, meningkatkan produktivitas, meningkatkan kualitas kode, dan memberikan lebih cepat. Untuk informasi lebih lanjut, lihat Manfaat pengiriman berkelanjutan. CD juga dapat berarti penerapan berkelanjutan. Untuk informasi selengkapnya, lihat Continuous Delivery vs Continuous Deployment.

CV

Lihat visi komputer.

D

data saat istirahat

Data yang stasioner di jaringan Anda, seperti data yang ada di penyimpanan.

klasifikasi data

Proses untuk mengidentifikasi dan mengkategorikan data dalam jaringan Anda berdasarkan kekritisan dan sensitivitasnya. Ini adalah komponen penting dari setiap strategi manajemen risiko keamanan siber karena membantu Anda menentukan perlindungan dan kontrol retensi yang tepat untuk data. Klasifikasi data adalah komponen pilar keamanan dalam AWS Well-Architected Framework. Untuk informasi selengkapnya, lihat Klasifikasi data.

penyimpangan data

Variasi yang berarti antara data produksi dan data yang digunakan untuk melatih model ML, atau perubahan yang berarti dalam data input dari waktu ke waktu. Penyimpangan data dapat mengurangi kualitas, akurasi, dan keadilan keseluruhan dalam prediksi model ML.

data dalam transit

Data yang aktif bergerak melalui jaringan Anda, seperti antara sumber daya jaringan.

jala data

Kerangka arsitektur yang menyediakan kepemilikan data terdistribusi dan terdesentralisasi dengan manajemen dan tata kelola terpusat.

minimalisasi data

Prinsip pengumpulan dan pemrosesan hanya data yang sangat diperlukan. Mempraktikkan minimalisasi data AWS Cloud dapat mengurangi risiko privasi, biaya, dan jejak karbon analitik Anda.

perimeter data

Satu set pagar pembatas pencegahan di AWS lingkungan Anda yang membantu memastikan bahwa hanya identitas tepercaya yang mengakses sumber daya tepercaya dari jaringan yang diharapkan. Untuk informasi selengkapnya, lihat Membangun perimeter data pada AWS.

prapemrosesan data

Untuk mengubah data mentah menjadi format yang mudah diuraikan oleh model ML Anda. Preprocessing data dapat berarti menghapus kolom atau baris tertentu dan menangani nilai yang hilang, tidak konsisten, atau duplikat.

asal data

Proses melacak asal dan riwayat data sepanjang siklus hidupnya, seperti bagaimana data dihasilkan, ditransmisikan, dan disimpan.

subjek data

Individu yang datanya dikumpulkan dan diproses.

gudang data

Sistem manajemen data yang mendukung intelijen bisnis, seperti analitik. Gudang data biasanya berisi sejumlah besar data historis, dan biasanya digunakan untuk kueri dan analisis.

bahasa definisi database (DDL)

Pernyataan atau perintah untuk membuat atau memodifikasi struktur tabel dan objek dalam database.

bahasa manipulasi basis data (DHTML)

Pernyataan atau perintah untuk memodifikasi (memasukkan, memperbarui, dan menghapus) informasi dalam database.

DDL

Lihat bahasa definisi database.

ansambel yang dalam

Untuk menggabungkan beberapa model pembelajaran mendalam untuk prediksi. Anda dapat menggunakan ansambel dalam untuk mendapatkan prediksi yang lebih akurat atau untuk memperkirakan ketidakpastian dalam prediksi.

pembelajaran mendalam

Subbidang ML yang menggunakan beberapa lapisan jaringan saraf tiruan untuk mengidentifikasi pemetaan antara data input dan variabel target yang diinginkan.

defense-in-depth

Pendekatan keamanan informasi di mana serangkaian mekanisme dan kontrol keamanan dilapisi dengan cermat di seluruh jaringan komputer untuk melindungi kerahasiaan, integritas, dan ketersediaan jaringan dan data di dalamnya. Saat Anda mengadopsi strategi ini AWS, Anda menambahkan beberapa kontrol pada lapisan AWS Organizations struktur yang berbeda untuk membantu mengamankan sumber daya. Misalnya, defense-in-depth pendekatan mungkin menggabungkan otentikasi multi-faktor, segmentasi jaringan, dan enkripsi.

administrator yang didelegasikan

Di AWS Organizations, layanan yang kompatibel dapat mendaftarkan akun AWS anggota untuk mengelola akun organisasi dan mengelola izin untuk layanan tersebut. Akun ini disebut administrator yang didelegasikan untuk layanan itu. Untuk informasi selengkapnya dan daftar layanan yang kompatibel, lihat Layanan yang berfungsi dengan AWS Organizations AWS Organizations dokumentasi.

deployment

Proses pembuatan aplikasi, fitur baru, atau perbaikan kode tersedia di lingkungan target. Deployment melibatkan penerapan perubahan dalam basis kode dan kemudian membangun dan menjalankan basis kode itu di lingkungan aplikasi.

lingkungan pengembangan

Lihat lingkungan.

kontrol detektif

Kontrol keamanan yang dirancang untuk mendeteksi, mencatat, dan memperingatkan setelah suatu peristiwa terjadi. Kontrol ini adalah garis pertahanan kedua, memperingatkan Anda tentang peristiwa keamanan yang melewati kontrol pencegahan yang ada. Untuk informasi selengkapnya, lihat Kontrol Detektif dalam Menerapkan kontrol keamanan pada. AWS

pemetaan aliran nilai pengembangan (DVSM)

Sebuah proses yang digunakan untuk mengidentifikasi dan memprioritaskan kendala yang mempengaruhi kecepatan dan kualitas dalam siklus hidup pengembangan perangkat lunak. DVSM memperluas proses pemetaan aliran nilai yang awalnya dirancang untuk praktik

manufaktur ramping. Ini berfokus pada langkah-langkah dan tim yang diperlukan untuk menciptakan dan memindahkan nilai melalui proses pengembangan perangkat lunak.

kembar digital

Representasi virtual dari sistem dunia nyata, seperti bangunan, pabrik, peralatan industri, atau jalur produksi. Kembar digital mendukung pemeliharaan prediktif, pemantauan jarak jauh, dan optimalisasi produksi.

tabel dimensi

Dalam skema bintang, tabel yang lebih kecil yang berisi atribut data tentang data kuantitatif dalam tabel fakta. Atribut tabel dimensi biasanya bidang teks atau angka diskrit yang berperilaku seperti teks. Atribut ini biasanya digunakan untuk pembatasan kueri, pemfilteran, dan pelabelan set hasil.

musibah

Peristiwa yang mencegah beban kerja atau sistem memenuhi tujuan bisnisnya di lokasi utama yang digunakan. Peristiwa ini dapat berupa bencana alam, kegagalan teknis, atau akibat dari tindakan manusia, seperti kesalahan konfigurasi yang tidak disengaja atau serangan malware.

pemulihan bencana (DR)

Strategi dan proses yang Anda gunakan untuk meminimalkan downtime dan kehilangan data yang disebabkan oleh bencana. Untuk informasi selengkapnya, lihat Disaster Recovery of Workloads on AWS: Recovery in the Cloud in the AWS Well-Architected Framework.

DML~

Lihat bahasa manipulasi database.

desain berbasis domain

Pendekatan untuk mengembangkan sistem perangkat lunak yang kompleks dengan menghubungkan komponennya ke domain yang berkembang, atau tujuan bisnis inti, yang dilayani oleh setiap komponen. Konsep ini diperkenalkan oleh Eric Evans dalam bukunya, Domain-Driven Design: Tackling Complexity in the Heart of Software (Boston: Addison-Wesley Professional, 2003). Untuk informasi tentang cara menggunakan desain berbasis domain dengan pola gambar pencekik, lihat Memodernisasi layanan web Microsoft ASP.NET (ASMX) lama secara bertahap menggunakan container dan Amazon API Gateway.

DR

Lihat pemulihan bencana.

deteksi drift

Melacak penyimpangan dari konfigurasi dasar. Misalnya, Anda dapat menggunakan AWS CloudFormation untuk mendeteksi penyimpangan dalam sumber daya sistem, atau Anda dapat menggunakannya AWS Control Tower untuk mendeteksi perubahan di landing zone yang mungkin memengaruhi kepatuhan terhadap persyaratan tata kelola.

DVSM

Lihat pemetaan aliran nilai pengembangan.

E

EDA

Lihat analisis data eksplorasi.

komputasi tepi

Teknologi yang meningkatkan daya komputasi untuk perangkat pintar di tepi jaringan IoT. Jika dibandingkan dengan komputasi awan, komputasi tepi dapat mengurangi latensi komunikasi dan meningkatkan waktu respons.

enkripsi

Proses komputasi yang mengubah data plaintext, yang dapat dibaca manusia, menjadi ciphertext.

kunci enkripsi

String kriptografi dari bit acak yang dihasilkan oleh algoritma enkripsi. Panjang kunci dapat bervariasi, dan setiap kunci dirancang agar tidak dapat diprediksi dan unik.

endianness

Urutan byte disimpan dalam memori komputer. Sistem big-endian menyimpan byte paling signifikan terlebih dahulu. Sistem little-endian menyimpan byte paling tidak signifikan terlebih dahulu.

titik akhir

Lihat titik akhir layanan.

layanan endpoint

Layanan yang dapat Anda host di cloud pribadi virtual (VPC) untuk dibagikan dengan pengguna lain. Anda dapat membuat layanan endpoint dengan AWS PrivateLink dan memberikan izin kepada prinsipal lain Akun AWS atau ke AWS Identity and Access Management (IAM). Akun atau prinsipal ini dapat terhubung ke layanan endpoint Anda secara pribadi dengan membuat titik akhir VPC antarmuka. Untuk informasi selengkapnya, lihat Membuat layanan titik akhir di dokumentasi Amazon Virtual Private Cloud (Amazon VPC).

perencanaan sumber daya perusahaan (ERP)

Sistem yang mengotomatiskan dan mengelola proses bisnis utama (seperti akuntansi, MES, dan manajemen proyek) untuk suatu perusahaan.

enkripsi amplop

Proses mengenkripsi kunci enkripsi dengan kunci enkripsi lain. Untuk informasi selengkapnya, lihat Enkripsi amplop dalam dokumentasi AWS Key Management Service (AWS KMS).

lingkungan

Sebuah contoh dari aplikasi yang sedang berjalan. Berikut ini adalah jenis lingkungan yang umum dalam komputasi awan:

- Development Environment — Sebuah contoh dari aplikasi yang berjalan yang hanya tersedia untuk tim inti yang bertanggung jawab untuk memelihara aplikasi. Lingkungan pengembangan digunakan untuk menguji perubahan sebelum mempromosikannya ke lingkungan atas. Jenis lingkungan ini kadang-kadang disebut sebagai lingkungan pengujian.
- lingkungan yang lebih rendah — Semua lingkungan pengembangan untuk aplikasi, seperti yang digunakan untuk build awal dan pengujian.
- lingkungan produksi — Sebuah contoh dari aplikasi yang berjalan yang pengguna akhir dapat mengakses. Dalam pipa CI/CD, lingkungan produksi adalah lingkungan penyebaran terakhir.
- lingkungan atas — Semua lingkungan yang dapat diakses oleh pengguna selain tim pengembangan inti. Ini dapat mencakup lingkungan produksi, lingkungan praproduksi, dan lingkungan untuk pengujian penerimaan pengguna.

epik

Dalam metodologi tangkas, kategori fungsional yang membantu mengatur dan memprioritaskan pekerjaan Anda. Epik memberikan deskripsi tingkat tinggi tentang persyaratan dan tugas

implementasi. Misalnya, epos keamanan AWS CAF mencakup manajemen identitas dan akses, kontrol detektif, keamanan infrastruktur, perlindungan data, dan respons insiden. Untuk informasi selengkapnya tentang epos dalam strategi AWS migrasi, lihat panduan implementasi program.

ERP

Lihat perencanaan sumber daya perusahaan.

analisis data eksplorasi (EDA)

Proses menganalisis dataset untuk memahami karakteristik utamanya. Anda mengumpulkan atau mengumpulkan data dan kemudian melakukan penyelidikan awal untuk menemukan pola, mendeteksi anomali, dan memeriksa asumsi. EDA dilakukan dengan menghitung statistik ringkasan dan membuat visualisasi data.

F

tabel fakta

Tabel tengah dalam skema bintang. Ini menyimpan data kuantitatif tentang operasi bisnis. Biasanya, tabel fakta berisi dua jenis kolom: kolom yang berisi ukuran dan yang berisi kunci asing ke tabel dimensi.

gagal cepat

Filosofi yang menggunakan pengujian yang sering dan bertahap untuk mengurangi siklus hidup pengembangan. Ini adalah bagian penting dari pendekatan tangkas.

batas isolasi kesalahan

Dalam AWS Cloud, batas seperti Availability Zone, Wilayah AWS, control plane, atau data plane yang membatasi efek kegagalan dan membantu meningkatkan ketahanan beban kerja. Untuk informasi selengkapnya, lihat Batas Isolasi AWS Kesalahan.

cabang fitur

Lihat cabang.

fitur

Data input yang Anda gunakan untuk membuat prediksi. Misalnya, dalam konteks manufaktur, fitur bisa berupa gambar yang diambil secara berkala dari lini manufaktur.

pentingnya fitur

Seberapa signifikan fitur untuk prediksi model. Ini biasanya dinyatakan sebagai skor numerik yang dapat dihitung melalui berbagai teknik, seperti Shapley Additive Explanations (SHAP) dan gradien terintegrasi. Untuk informasi lebih lanjut, lihat Interpretabilitas model pembelajaran mesin dengan:.AWS

transformasi fitur

Untuk mengoptimalkan data untuk proses ML, termasuk memperkaya data dengan sumber tambahan, menskalakan nilai, atau mengekstrak beberapa set informasi dari satu bidang data. Hal ini memungkinkan model ML untuk mendapatkan keuntungan dari data. Misalnya, jika Anda memecah tanggal “2021-05-27 00:15:37" menjadi “2021", “Mei”, “Kamis”, dan “15", Anda dapat membantu algoritme pembelajaran mempelajari pola bernuansa yang terkait dengan komponen data yang berbeda.

FGAC

Lihat kontrol akses berbutir halus.

kontrol akses berbutir halus (FGAC)

Penggunaan beberapa kondisi untuk mengizinkan atau menolak permintaan akses.

migrasi flash-cut

Metode migrasi database yang menggunakan replikasi data berkelanjutan melalui pengambilan data perubahan untuk memigrasikan data dalam waktu sesingkat mungkin, alih-alih menggunakan pendekatan bertahap. Tujuannya adalah untuk menjaga downtime seminimal mungkin.

G

pemblokiran geografis

Lihat pembatasan geografis.

pembatasan geografis (pemblokiran geografis)

Di Amazon CloudFront, opsi untuk mencegah pengguna di negara tertentu mengakses distribusi konten. Anda dapat menggunakan daftar izinkan atau daftar blokir untuk menentukan negara yang disetujui dan dilarang. Untuk informasi selengkapnya, lihat Membatasi distribusi geografis konten Anda dalam dokumentasi. CloudFront

Alur kerja Gitflow

Pendekatan di mana lingkungan bawah dan atas menggunakan cabang yang berbeda dalam repositori kode sumber. Alur kerja Gitflow dianggap warisan, dan alur kerja berbasis batang adalah pendekatan modern yang lebih disukai.

strategi greenfield

Tidak adanya infrastruktur yang ada di lingkungan baru. Saat mengadopsi strategi greenfield untuk arsitektur sistem, Anda dapat memilih semua teknologi baru tanpa batasan kompatibilitas dengan infrastruktur yang ada, juga dikenal sebagai brownfield. Jika Anda memperluas infrastruktur yang ada, Anda dapat memadukan strategi brownfield dan greenfield.

pagar pembatas

Aturan tingkat tinggi yang membantu mengatur sumber daya, kebijakan, dan kepatuhan di seluruh unit organisasi (OU). Pagar pembatas preventif menegakkan kebijakan untuk memastikan keselarasan dengan standar kepatuhan. Mereka diimplementasikan dengan menggunakan kebijakan kontrol layanan dan batas izin IAM. Detective guardrails mendeteksi pelanggaran kebijakan dan masalah kepatuhan, dan menghasilkan peringatan untuk remediasi. Mereka diimplementasikan dengan menggunakan AWS Config, AWS Security Hub, Amazon GuardDuty AWS Trusted Advisor, Amazon Inspector, dan pemeriksaan khusus AWS Lambda .

H

HA

Lihat ketersediaan tinggi.

migrasi database heterogen

Memigrasi database sumber Anda ke database target yang menggunakan mesin database yang berbeda (misalnya, Oracle ke Amazon Aurora). Migrasi heterogen biasanya merupakan bagian dari upaya arsitektur ulang, dan mengubah skema dapat menjadi tugas yang kompleks. AWS menyediakan AWS SCT yang membantu dengan konversi skema.

ketersediaan tinggi (HA)

Kemampuan beban kerja untuk beroperasi terus menerus, tanpa intervensi, jika terjadi tantangan atau bencana. Sistem HA dirancang untuk gagal secara otomatis, secara konsisten memberikan kinerja berkualitas tinggi, dan menangani beban dan kegagalan yang berbeda dengan dampak kinerja minimal.

modernisasi sejarawan

Pendekatan yang digunakan untuk memodernisasi dan meningkatkan sistem teknologi operasional (OT) untuk melayani kebutuhan industri manufaktur dengan lebih baik. Sejarawan adalah jenis database yang digunakan untuk mengumpulkan dan menyimpan data dari berbagai sumber di pabrik.

migrasi database homogen

Memigrasi database sumber Anda ke database target yang berbagi mesin database yang sama (misalnya, Microsoft SQL Server ke Amazon RDS for SQL Server). Migrasi homogen biasanya merupakan bagian dari upaya rehosting atau replatforming. Anda dapat menggunakan utilitas database asli untuk memigrasi skema.

data panas

Data yang sering diakses, seperti data real-time atau data translasi terbaru. Data ini biasanya memerlukan tingkat atau kelas penyimpanan berkinerja tinggi untuk memberikan respons kueri yang cepat.

perbaikan terbaru

Perbaikan mendesak untuk masalah kritis dalam lingkungan produksi. Karena urgensinya, perbaikan terbaru biasanya dibuat di luar alur kerja DevOps rilis biasa.

periode hypercare

Segera setelah cutover, periode waktu ketika tim migrasi mengelola dan memantau aplikasi yang dimigrasi di cloud untuk mengatasi masalah apa pun. Biasanya, periode ini panjangnya 1-4 hari. Pada akhir periode hypercare, tim migrasi biasanya mentransfer tanggung jawab untuk aplikasi ke tim operasi cloud.

I

IAc

Lihat infrastruktur sebagai kode.

kebijakan berbasis identitas

Kebijakan yang dilampirkan pada satu atau beberapa prinsip IAM yang mendefinisikan izin mereka dalam lingkungan. AWS Cloud

aplikasi idle

Aplikasi yang memiliki penggunaan CPU dan memori rata-rata antara 5 dan 20 persen selama periode 90 hari. Dalam proyek migrasi, adalah umum untuk menghentikan aplikasi ini atau mempertahankannya di tempat.

IIoT

Lihat Internet of Things industri.

infrastruktur yang tidak dapat diubah

Model yang menyebarkan infrastruktur baru untuk beban kerja produksi alih-alih memperbarui, menambal, atau memodifikasi infrastruktur yang ada. Infrastruktur yang tidak dapat diubah secara inheren lebih konsisten, andal, dan dapat diprediksi daripada infrastruktur yang dapat berubah. Untuk informasi selengkapnya, lihat praktik terbaik Deploy using immutable infrastructure di AWS Well-Architected Framework.

masuk (masuknya) VPC

Dalam arsitektur AWS multi-akun, VPC yang menerima, memeriksa, dan merutekan koneksi jaringan dari luar aplikasi. Arsitektur Referensi AWS Keamanan merekomendasikan pengaturan akun Jaringan Anda dengan VPC masuk, keluar, dan inspeksi untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

migrasi inkremental

Strategi cutover di mana Anda memigrasikan aplikasi Anda dalam bagian-bagian kecil alih-alih melakukan satu cutover penuh. Misalnya, Anda mungkin hanya memindahkan beberapa layanan mikro atau pengguna ke sistem baru pada awalnya. Setelah Anda memverifikasi bahwa semuanya berfungsi dengan baik, Anda dapat secara bertahap memindahkan layanan mikro atau pengguna tambahan hingga Anda dapat menonaktifkan sistem lama Anda. Strategi ini mengurangi risiko yang terkait dengan migrasi besar.

Industri 4.0

Sebuah istilah yang diperkenalkan oleh Klaus Schwab pada tahun 2016 untuk merujuk pada modernisasi proses manufaktur melalui kemajuan dalam konektivitas, data real-time, otomatisasi, analitik, dan AI/ML.

infrastruktur

Semua sumber daya dan aset yang terkandung dalam lingkungan aplikasi.

infrastruktur sebagai kode (IAc)

Proses penyediaan dan pengelolaan infrastruktur aplikasi melalui satu set file konfigurasi. IAc dirancang untuk membantu Anda memusatkan manajemen infrastruktur, menstandarisasi sumber daya, dan menskalakan dengan cepat sehingga lingkungan baru dapat diulang, andal, dan konsisten.

Internet of Things industri (IIoT)

Penggunaan sensor dan perangkat yang terhubung ke internet di sektor industri, seperti manufaktur, energi, otomotif, perawatan kesehatan, ilmu kehidupan, dan pertanian. Untuk informasi selengkapnya, lihat Membangun strategi transformasi digital Internet of Things (IIoT) industri.

inspeksi VPC

Dalam arsitektur AWS multi-akun, VPC terpusat yang mengelola inspeksi lalu lintas jaringan antara VPC (dalam hal yang sama atau berbeda Wilayah AWS), internet, dan jaringan lokal. Arsitektur Referensi AWS Keamanan merekomendasikan pengaturan akun Jaringan Anda dengan VPC masuk, keluar, dan inspeksi untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

Internet of Things (IoT)

Jaringan objek fisik yang terhubung dengan sensor atau prosesor tertanam yang berkomunikasi dengan perangkat dan sistem lain melalui internet atau melalui jaringan komunikasi lokal. Untuk informasi selengkapnya, lihat Apa itu IoT?

interpretabilitas

Karakteristik model pembelajaran mesin yang menggambarkan sejauh mana manusia dapat memahami bagaimana prediksi model bergantung pada inputnya. Untuk informasi selengkapnya, lihat Interpretabilitas model pembelajaran mesin dengan AWS.

IoT

Lihat Internet of Things.

Perpustakaan informasi TI (ITIL)

Serangkaian praktik terbaik untuk memberikan layanan TI dan menyelaraskan layanan ini dengan persyaratan bisnis. ITIL menyediakan dasar untuk ITSM.

Manajemen layanan TI (ITSM)

Kegiatan yang terkait dengan merancang, menerapkan, mengelola, dan mendukung layanan TI untuk suatu organisasi. Untuk informasi tentang mengintegrasikan operasi cloud dengan alat ITSM, lihat panduan integrasi operasi.

ITIL

Lihat perpustakaan informasi TI.

ITSM

Lihat manajemen layanan TI.

L

kontrol akses berbasis label (LBAC)

Implementasi kontrol akses wajib (MAC) di mana pengguna dan data itu sendiri masing-masing secara eksplisit diberi nilai label keamanan. Persimpangan antara label keamanan pengguna dan label keamanan data menentukan baris dan kolom mana yang dapat dilihat oleh pengguna.

landing zone

Landing zone adalah AWS lingkungan multi-akun yang dirancang dengan baik yang dapat diskalakan dan aman. Ini adalah titik awal dari mana organisasi Anda dapat dengan cepat meluncurkan dan menyebarkan beban kerja dan aplikasi dengan percaya diri dalam lingkungan keamanan dan infrastruktur mereka. Untuk informasi selengkapnya tentang zona pendaratan, lihat Menyiapkan lingkungan multi-akun AWS yang aman dan dapat diskalakan.

migrasi besar

Migrasi 300 atau lebih server.

LBAC

Lihat kontrol akses berbasis label.

hak istimewa paling sedikit

Praktik keamanan terbaik untuk memberikan izin minimum yang diperlukan untuk melakukan tugas. Untuk informasi selengkapnya, lihat Menerapkan izin hak istimewa terkecil dalam dokumentasi IAM.

angkat dan geser

Lihat 7 Rs.

sistem endian kecil

Sebuah sistem yang menyimpan byte paling tidak signifikan terlebih dahulu. Lihat juga endianness.

lingkungan yang lebih rendah

Lihat lingkungan.

M

pembelajaran mesin (ML)

Jenis kecerdasan buatan yang menggunakan algoritma dan teknik untuk pengenalan pola dan pembelajaran. ML menganalisis dan belajar dari data yang direkam, seperti data Internet of Things (IoT), untuk menghasilkan model statistik berdasarkan pola. Untuk informasi selengkapnya, lihat Machine Learning.

cabang utama

Lihat cabang.

malware

Perangkat lunak yang dirancang untuk membahayakan keamanan atau privasi komputer. Malware dapat mengganggu sistem komputer, membocorkan informasi sensitif, atau mendapatkan akses yang tidak sah. Contoh malware termasuk virus, worm, ransomware, Trojan horse, spyware, dan keyloggers.

layanan terkelola

Layanan AWS yang AWS mengoperasikan lapisan infrastruktur, sistem operasi, dan platform, dan Anda mengakses titik akhir untuk menyimpan dan mengambil data. Amazon Simple Storage Service (Amazon S3) dan Amazon DynamoDB adalah contoh layanan terkelola. Ini juga dikenal sebagai layanan abstrak.

sistem eksekusi manufaktur (MES)

Sistem perangkat lunak untuk melacak, memantau, mendokumentasikan, dan mengendalikan proses produksi yang mengubah bahan baku menjadi produk jadi di lantai toko.

PETA

Lihat Program Percepatan Migrasi.

mekanisme

Proses lengkap di mana Anda membuat alat, mendorong adopsi alat, dan kemudian memeriksa hasilnya untuk melakukan penyesuaian. Mekanisme adalah siklus yang memperkuat dan meningkatkan dirinya sendiri saat beroperasi. Untuk informasi lebih lanjut, lihat Membangun mekanisme di AWS Well-Architected Framework.

akun anggota

Semua Akun AWS selain akun manajemen yang merupakan bagian dari organisasi di AWS Organizations. Akun dapat menjadi anggota dari hanya satu organisasi pada suatu waktu.

MES

Lihat sistem eksekusi manufaktur.

Transportasi Telemetri Antrian Pesan (MQTT)

Protokol komunikasi ringan machine-to-machine (M2M), berdasarkan pola terbitkan/berlangganan, untuk perangkat IoT yang dibatasi sumber daya.

layanan mikro

Layanan kecil dan independen yang berkomunikasi melalui API yang terdefinisi dengan baik dan biasanya dimiliki oleh tim kecil yang mandiri. Misalnya, sistem asuransi mungkin mencakup layanan mikro yang memetakan kemampuan bisnis, seperti penjualan atau pemasaran, atau subdomain, seperti pembelian, klaim, atau analitik. Manfaat layanan mikro termasuk kelincahan, penskalaan yang fleksibel, penyebaran yang mudah, kode yang dapat digunakan kembali, dan ketahanan. Untuk informasi selengkapnya, lihat Mengintegrasikan layanan mikro dengan menggunakan layanan tanpa AWS server.

arsitektur microservices

Pendekatan untuk membangun aplikasi dengan komponen independen yang menjalankan setiap proses aplikasi sebagai layanan mikro. Layanan mikro ini berkomunikasi melalui antarmuka yang terdefinisi dengan baik dengan menggunakan API ringan. Setiap layanan mikro dalam arsitektur ini dapat diperbarui, digunakan, dan diskalakan untuk memenuhi permintaan fungsi tertentu dari suatu aplikasi. Untuk informasi selengkapnya, lihat Menerapkan layanan mikro di AWS.

Program Percepatan Migrasi (MAP)

AWS Program yang menyediakan dukungan konsultasi, pelatihan, dan layanan untuk membantu organisasi membangun fondasi operasional yang kuat untuk pindah ke cloud, dan untuk membantu mengimbangi biaya awal migrasi. MAP mencakup metodologi migrasi untuk mengeksekusi migrasi lama dengan cara metodis dan seperangkat alat untuk mengotomatisasi dan mempercepat skenario migrasi umum.

migrasi dalam skala

Proses memindahkan sebagian besar portofolio aplikasi ke cloud dalam gelombang, dengan lebih banyak aplikasi bergerak pada tingkat yang lebih cepat di setiap gelombang. Fase ini menggunakan praktik terbaik dan pelajaran yang dipetik dari fase sebelumnya untuk mengimplementasikan pabrik migrasi tim, alat, dan proses untuk merampingkan migrasi beban kerja melalui otomatisasi dan pengiriman tangkas. Ini adalah fase ketiga dari strategi AWS migrasi.

pabrik migrasi

Tim lintas fungsi yang merampingkan migrasi beban kerja melalui pendekatan otomatis dan gesit. Tim pabrik migrasi biasanya mencakup operasi, analis dan pemilik bisnis, insinyur migrasi, pengembang, dan DevOps profesional yang bekerja di sprint. Antara 20 dan 50 persen portofolio aplikasi perusahaan terdiri dari pola berulang yang dapat dioptimalkan dengan pendekatan pabrik. Untuk informasi selengkapnya, lihat diskusi tentang pabrik migrasi dan panduan Pabrik Migrasi Cloud di kumpulan konten ini.

metadata migrasi

Informasi tentang aplikasi dan server yang diperlukan untuk menyelesaikan migrasi. Setiap pola migrasi memerlukan satu set metadata migrasi yang berbeda. Contoh metadata migrasi termasuk subnet target, grup keamanan, dan akun. AWS

pola migrasi

Tugas migrasi berulang yang merinci strategi migrasi, tujuan migrasi, dan aplikasi atau layanan migrasi yang digunakan. Contoh: Rehost migrasi ke Amazon EC2 AWS dengan Layanan Migrasi Aplikasi.

Penilaian Portofolio Migrasi (MPA)

Alat online yang menyediakan informasi untuk memvalidasi kasus bisnis untuk bermigrasi ke. AWS Cloud MPA menyediakan penilaian portofolio terperinci (ukuran kanan server, harga,

perbandingan TCO, analisis biaya migrasi) serta perencanaan migrasi (analisis data aplikasi dan pengumpulan data, pengelompokan aplikasi, prioritas migrasi, dan perencanaan gelombang). Alat MPA (memerlukan login) tersedia gratis untuk semua AWS konsultan dan konsultan APN Partner.

Penilaian Kesiapan Migrasi (MRA)

Proses mendapatkan wawasan tentang status kesiapan cloud organisasi, mengidentifikasi kekuatan dan kelemahan, dan membangun rencana aksi untuk menutup kesenjangan yang diidentifikasi, menggunakan CAF. AWS Untuk informasi selengkapnya, lihat panduan kesiapan migrasi. MRA adalah tahap pertama dari strategi AWS migrasi.

strategi migrasi

Pendekatan yang digunakan untuk memigrasikan beban kerja ke file. AWS Cloud Untuk informasi lebih lanjut, lihat entri 7 Rs di glosarium ini dan lihat Memobilisasi organisasi Anda untuk mempercepat migrasi skala besar.

ML

Lihat pembelajaran mesin.

modernisasi

Mengubah aplikasi usang (warisan atau monolitik) dan infrastrukturnya menjadi sistem yang gesit, elastis, dan sangat tersedia di cloud untuk mengurangi biaya, mendapatkan efisiensi, dan memanfaatkan inovasi. Untuk informasi selengkapnya, lihat Strategi untuk memodernisasi aplikasi di. AWS Cloud

penilaian kesiapan modernisasi

Evaluasi yang membantu menentukan kesiapan modernisasi aplikasi organisasi; mengidentifikasi manfaat, risiko, dan dependensi; dan menentukan seberapa baik organisasi dapat mendukung keadaan masa depan aplikasi tersebut. Hasil penilaian adalah cetak biru arsitektur target, peta jalan yang merinci fase pengembangan dan tonggak untuk proses modernisasi, dan rencana aksi untuk mengatasi kesenjangan yang diidentifikasi. Untuk informasi lebih lanjut, lihat Mengevaluasi kesiapan modernisasi untuk aplikasi di. AWS Cloud

aplikasi monolitik (monolit)

Aplikasi yang berjalan sebagai layanan tunggal dengan proses yang digabungkan secara ketat. Aplikasi monolitik memiliki beberapa kelemahan. Jika satu fitur aplikasi mengalami lonjakan permintaan, seluruh arsitektur harus diskalakan. Menambahkan atau meningkatkan fitur aplikasi monolitik juga menjadi lebih kompleks ketika basis kode tumbuh. Untuk mengatasi masalah ini,

Anda dapat menggunakan arsitektur microservices. Untuk informasi lebih lanjut, lihat Mengurai monolit menjadi layanan mikro.

MPA

Lihat Penilaian Portofolio Migrasi.

MQTT

Lihat Transportasi Telemetri Antrian Pesan.

klasifikasi multiclass

Sebuah proses yang membantu menghasilkan prediksi untuk beberapa kelas (memprediksi satu dari lebih dari dua hasil). Misalnya, model ML mungkin bertanya "Apakah produk ini buku, mobil, atau telepon?" atau "Kategori produk mana yang paling menarik bagi pelanggan ini?"

infrastruktur yang bisa berubah

Model yang memperbarui dan memodifikasi infrastruktur yang ada untuk beban kerja produksi. Untuk meningkatkan konsistensi, keandalan, dan prediktabilitas, AWS Well-Architected Framework merekomendasikan penggunaan infrastruktur yang tidak dapat diubah sebagai praktik terbaik.

O

OAC

Lihat kontrol akses asal.

OAI

Lihat identitas akses asal.

OCM

Lihat manajemen perubahan organisasi.

migrasi offline

Metode migrasi di mana beban kerja sumber diturunkan selama proses migrasi. Metode ini melibatkan waktu henti yang diperpanjang dan biasanya digunakan untuk beban kerja kecil dan tidak kritis.

OI

Lihat integrasi operasi.

OLA

Lihat perjanjian tingkat operasional.

migrasi online

Metode migrasi di mana beban kerja sumber disalin ke sistem target tanpa diambil offline. Aplikasi yang terhubung ke beban kerja dapat terus berfungsi selama migrasi. Metode ini melibatkan waktu henti nol hingga minimal dan biasanya digunakan untuk beban kerja produksi yang kritis.

OPC-UA

Lihat Komunikasi Proses Terbuka - Arsitektur Terpadu.

Komunikasi Proses Terbuka - Arsitektur Terpadu (OPC-UA)

Protokol komunikasi machine-to-machine (M2M) untuk otomasi industri. OPC-UA menyediakan standar interoperabilitas dengan enkripsi data, otentikasi, dan skema otorisasi.

perjanjian tingkat operasional (OLA)

Perjanjian yang menjelaskan apa yang dijanjikan kelompok TI fungsional untuk diberikan satu sama lain, untuk mendukung perjanjian tingkat layanan (SLA).

Tinjauan Kesiapan Operasional (ORR)

Daftar pertanyaan dan praktik terbaik terkait yang membantu Anda memahami, mengevaluasi, mencegah, atau mengurangi ruang lingkup insiden dan kemungkinan kegagalan. Untuk informasi lebih lanjut, lihat Ulasan Kesiapan Operasional (ORR) dalam Kerangka Kerja Well-Architected AWS .

teknologi operasional (OT)

Sistem perangkat keras dan perangkat lunak yang bekerja dengan lingkungan fisik untuk mengendalikan operasi industri, peralatan, dan infrastruktur. Di bidang manufaktur, integrasi sistem OT dan teknologi informasi (TI) adalah fokus utama untuk transformasi Industri 4.0.

integrasi operasi (OI)

Proses modernisasi operasi di cloud, yang melibatkan perencanaan kesiapan, otomatisasi, dan integrasi. Untuk informasi selengkapnya, lihat panduan integrasi operasi.

jejak organisasi

Jejak yang dibuat oleh AWS CloudTrail itu mencatat semua peristiwa untuk semua Akun AWS dalam organisasi di AWS Organizations. Jejak ini dibuat di setiap Akun AWS bagian organisasi dan melacak aktivitas di setiap akun. Untuk informasi selengkapnya, lihat Membuat jejak untuk organisasi dalam CloudTrail dokumentasi.

manajemen perubahan organisasi (OCM)

Kerangka kerja untuk mengelola transformasi bisnis utama yang mengganggu dari perspektif orang, budaya, dan kepemimpinan. OCM membantu organisasi mempersiapkan, dan transisi ke, sistem dan strategi baru dengan mempercepat adopsi perubahan, mengatasi masalah transisi, dan mendorong perubahan budaya dan organisasi. Dalam strategi AWS migrasi, kerangka kerja ini disebut percepatan orang, karena kecepatan perubahan yang diperlukan dalam proyek adopsi cloud. Untuk informasi lebih lanjut, lihat panduan OCM.

kontrol akses asal (OAC)

Di CloudFront, opsi yang disempurnakan untuk membatasi akses untuk mengamankan konten Amazon Simple Storage Service (Amazon S3) Anda. OAC mendukung semua bucket S3 di semua Wilayah AWS, enkripsi sisi server dengan AWS KMS (SSE-KMS), dan dinamis dan permintaan ke bucket S3. `PUT` `DELETE`

identitas akses asal (OAI)

Di CloudFront, opsi untuk membatasi akses untuk mengamankan konten Amazon S3 Anda. Saat Anda menggunakan OAI, CloudFront buat prinsipal yang dapat diautentikasi oleh Amazon S3. Prinsipal yang diautentikasi dapat mengakses konten dalam bucket S3 hanya melalui distribusi tertentu. CloudFront Lihat juga OAC, yang menyediakan kontrol akses yang lebih terperinci dan ditingkatkan.

ORR

Lihat tinjauan kesiapan operasional.

OT

Lihat teknologi operasional.

keluar (jalan keluar) VPC

Dalam arsitektur AWS multi-akun, VPC yang menangani koneksi jaringan yang dimulai dari dalam aplikasi. Arsitektur Referensi AWS Keamanan merekomendasikan pengaturan akun Jaringan

Anda dengan VPC masuk, keluar, dan inspeksi untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

P

batas izin

Kebijakan manajemen IAM yang dilampirkan pada prinsipal IAM untuk menetapkan izin maksimum yang dapat dimiliki pengguna atau peran. Untuk informasi selengkapnya, lihat Batas izin dalam dokumentasi IAM.

Informasi Identifikasi Pribadi (PII)

Informasi yang, jika dilihat secara langsung atau dipasangkan dengan data terkait lainnya, dapat digunakan untuk menyimpulkan identitas individu secara wajar. Contoh PII termasuk nama, alamat, dan informasi kontak.

PII

Lihat informasi yang dapat diidentifikasi secara pribadi.

buku pedoman

Serangkaian langkah yang telah ditentukan sebelumnya yang menangkap pekerjaan yang terkait dengan migrasi, seperti mengirimkan fungsi operasi inti di cloud. Buku pedoman dapat berupa skrip, runbook otomatis, atau ringkasan proses atau langkah-langkah yang diperlukan untuk mengoperasikan lingkungan modern Anda.

PLC

Lihat pengontrol logika yang dapat diprogram.

PLM

Lihat manajemen siklus hidup produk.

kebijakan

Objek yang dapat menentukan izin (lihat kebijakan berbasis identitas), menentukan kondisi akses (lihat kebijakanberbasis sumber daya), atau menentukan izin maksimum untuk semua akun dalam organisasi di (lihat kebijakan kontrol layanan). AWS Organizations

persistensi poliglot

Secara independen memilih teknologi penyimpanan data microservice berdasarkan pola akses data dan persyaratan lainnya. Jika layanan mikro Anda memiliki teknologi penyimpanan data yang sama, mereka dapat menghadapi tantangan implementasi atau mengalami kinerja yang buruk. Layanan mikro lebih mudah diimplementasikan dan mencapai kinerja dan skalabilitas yang lebih baik jika mereka menggunakan penyimpanan data yang paling sesuai dengan kebutuhan mereka. Untuk informasi selengkapnya, lihat Mengaktifkan persistensi data di layanan mikro.

penilaian portofolio

Proses menemukan, menganalisis, dan memprioritaskan portofolio aplikasi untuk merencanakan migrasi. Untuk informasi selengkapnya, lihat Mengevaluasi kesiapan migrasi.

predikat

Kondisi kueri yang mengembalikan `true` atau`false`, biasanya terletak di `WHERE` klausa.

predikat pushdown

Teknik pengoptimalan kueri database yang menyaring data dalam kueri sebelum transfer. Ini mengurangi jumlah data yang harus diambil dan diproses dari database relasional, dan meningkatkan kinerja kueri.

kontrol preventif

Kontrol keamanan yang dirancang untuk mencegah suatu peristiwa terjadi. Kontrol ini adalah garis pertahanan pertama untuk membantu mencegah akses tidak sah atau perubahan yang tidak diinginkan ke jaringan Anda. Untuk informasi selengkapnya, lihat Kontrol pencegahan dalam Menerapkan kontrol keamanan pada. AWS

principal

Entitas AWS yang dapat melakukan tindakan dan mengakses sumber daya. Entitas ini biasanya merupakan pengguna root untuk Akun AWS, peran IAM, atau pengguna. Untuk informasi selengkapnya, lihat Prinsip dalam istilah dan konsep Peran dalam dokumentasi IAM.

Privasi oleh Desain

Pendekatan dalam rekayasa sistem yang memperhitungkan privasi di seluruh proses rekayasa.

zona host pribadi

Container yang menyimpan informasi tentang bagaimana Anda ingin Amazon Route 53 merespons kueri DNS untuk domain dan subdomainnya dalam satu atau beberapa VPC. Untuk

informasi selengkapnya, lihat Bekerja dengan zona yang dihosting pribadi di dokumentasi Route 53.

kontrol proaktif

Kontrol keamanan yang dirancang untuk mencegah penyebaran sumber daya yang tidak sesuai. Kontrol ini memindai sumber daya sebelum disediakan. Jika sumber daya tidak sesuai dengan kontrol, maka itu tidak disediakan. Untuk informasi selengkapnya, lihat panduan referensi Kontrol dalam AWS Control Tower dokumentasi dan lihat Kontrol proaktif dalam Menerapkan kontrol keamanan pada AWS.

manajemen siklus hidup produk (PLM)

Manajemen data dan proses untuk suatu produk di seluruh siklus hidupnya, mulai dari desain, pengembangan, dan peluncuran, melalui pertumbuhan dan kematangan, hingga penurunan dan penghapusan.

lingkungan produksi

Lihat lingkungan.

pengontrol logika yang dapat diprogram (PLC)

Di bidang manufaktur, komputer yang sangat andal dan mudah beradaptasi yang memantau mesin dan mengotomatiskan proses manufaktur.

pseudonimisasi

Proses penggantian pengidentifikasi pribadi dalam kumpulan data dengan nilai placeholder. Pseudonimisasi dapat membantu melindungi privasi pribadi. Data pseudonim masih dianggap sebagai data pribadi.

terbitkan/berlangganan (pub/sub)

Pola yang memungkinkan komunikasi asinkron antara layanan mikro untuk meningkatkan skalabilitas dan daya tanggap. Misalnya, dalam MES berbasis layanan mikro, layanan mikro dapat mempublikasikan pesan peristiwa ke saluran yang dapat berlangganan layanan mikro lainnya. Sistem dapat menambahkan layanan mikro baru tanpa mengubah layanan penerbitan.

Q

rencana kueri

Serangkaian langkah, seperti instruksi, yang digunakan untuk mengakses data dalam sistem database relasional SQL.

regresi rencana kueri

Ketika pengoptimal layanan database memilih rencana yang kurang optimal daripada sebelum perubahan yang diberikan ke lingkungan database. Hal ini dapat disebabkan oleh perubahan statistik, kendala, pengaturan lingkungan, pengikatan parameter kueri, dan pembaruan ke mesin database.

R

Matriks RACI

Lihat bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan (RACI).

ransomware

Perangkat lunak berbahaya yang dirancang untuk memblokir akses ke sistem komputer atau data sampai pembayaran dilakukan.

Matriks RASCI

Lihat bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan (RACI).

RCAC

Lihat kontrol akses baris dan kolom.

replika baca

Salinan database yang digunakan untuk tujuan read-only. Anda dapat merutekan kueri ke replika baca untuk mengurangi beban pada database utama Anda.

arsitek ulang

Lihat 7 Rs.

tujuan titik pemulihan (RPO)

Jumlah waktu maksimum yang dapat diterima sejak titik pemulihan data terakhir. Ini menentukan apa yang dianggap sebagai hilangnya data yang dapat diterima antara titik pemulihan terakhir dan gangguan layanan.

tujuan waktu pemulihan (RTO)

Penundaan maksimum yang dapat diterima antara gangguan layanan dan pemulihan layanan.

refactor

Lihat 7 Rs.

Wilayah

Kumpulan AWS sumber daya di wilayah geografis. Masing-masing Wilayah AWS terisolasi dan independen dari yang lain untuk memberikan toleransi kesalahan, stabilitas, dan ketahanan. Untuk informasi selengkapnya, lihat Menentukan Wilayah AWS akun yang dapat digunakan.

regresi

Teknik ML yang memprediksi nilai numerik. Misalnya, untuk memecahkan masalah “Berapa harga rumah ini akan dijual?” Model ML dapat menggunakan model regresi linier untuk memprediksi harga jual rumah berdasarkan fakta yang diketahui tentang rumah (misalnya, luas persegi).

rehost

Lihat 7 Rs.

melepaskan

Dalam proses penyebaran, tindakan mempromosikan perubahan pada lingkungan produksi.

memindahkan

Lihat 7 Rs.

memplatform ulang

Lihat 7 Rs.

pembelian kembali

Lihat 7 Rs.

ketahanan

Kemampuan aplikasi untuk melawan atau pulih dari gangguan. Ketersediaan tinggi dan pemulihan bencana adalah pertimbangan umum ketika merencanakan ketahanan di. AWS Cloud Untuk informasi lebih lanjut, lihat AWS Cloud Ketahanan.

kebijakan berbasis sumber daya

Kebijakan yang dilampirkan ke sumber daya, seperti bucket Amazon S3, titik akhir, atau kunci enkripsi. Jenis kebijakan ini menentukan prinsip mana yang diizinkan mengakses, tindakan yang didukung, dan kondisi lain yang harus dipenuhi.

matriks yang bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan (RACI)

Matriks yang mendefinisikan peran dan tanggung jawab untuk semua pihak yang terlibat dalam kegiatan migrasi dan operasi cloud. Nama matriks berasal dari jenis tanggung jawab yang didefinisikan dalam matriks: bertanggung jawab (R), akuntabel (A), dikonsultasikan (C), dan diinformasikan (I). Jenis dukungan (S) adalah opsional. Jika Anda menyertakan dukungan, matriks disebut matriks RASCI, dan jika Anda mengecualikannya, itu disebut matriks RACI.

kontrol responsif

Kontrol keamanan yang dirancang untuk mendorong remediasi efek samping atau penyimpangan dari garis dasar keamanan Anda. Untuk informasi selengkapnya, lihat Kontrol responsif dalam Menerapkan kontrol keamanan pada AWS.

melestarikan

Lihat 7 Rs.

pensiun

Lihat 7 Rs.

rotasi

Proses memperbarui rahasia secara berkala untuk membuatnya lebih sulit bagi penyerang untuk mengakses kredensil.

kontrol akses baris dan kolom (RCAC)

Penggunaan ekspresi SQL dasar dan fleksibel yang telah menetapkan aturan akses. RCAC terdiri dari izin baris dan topeng kolom.

RPO

Lihat tujuan titik pemulihan.

RTO

Lihat tujuan waktu pemulihan.

buku runbook

Satu set prosedur manual atau otomatis yang diperlukan untuk melakukan tugas tertentu. Ini biasanya dibangun untuk merampingkan operasi berulang atau prosedur dengan tingkat kesalahan yang tinggi.

D

SAML 2.0

Standar terbuka yang digunakan oleh banyak penyedia identitas (IdPs). Fitur ini memungkinkan sistem masuk tunggal gabungan (SSO), sehingga pengguna dapat masuk ke AWS Management Console atau memanggil operasi AWS API tanpa Anda harus membuat pengguna di IAM untuk semua orang di organisasi Anda. Untuk informasi lebih lanjut tentang federasi berbasis SAMP 2.0, lihat Tentang federasi berbasis SAMP 2.0 dalam dokumentasi IAM.

PENIPUAN

Lihat kontrol pengawasan dan akuisisi data.

SCP

Lihat kebijakan kontrol layanan.

Rahasia

Dalam AWS Secrets Manager, informasi rahasia atau terbatas, seperti kata sandi atau kredensi pengguna, yang Anda simpan dalam bentuk terenkripsi. Ini terdiri dari nilai rahasia dan metadata-nya. Nilai rahasia dapat berupa biner, string tunggal, atau beberapa string. Untuk informasi selengkapnya, lihat Apa yang ada di rahasia Secrets Manager? dalam dokumentasi Secrets Manager.

kontrol keamanan

Pagar pembatas teknis atau administratif yang mencegah, mendeteksi, atau mengurangi kemampuan pelaku ancaman untuk mengeksploitasi kerentanan keamanan. Ada empat jenis kontrol keamanan utama: preventif, detektif, responsif, dan proaktif.

pengerasan keamanan

Proses mengurangi permukaan serangan untuk membuatnya lebih tahan terhadap serangan. Ini dapat mencakup tindakan seperti menghapus sumber daya yang tidak lagi diperlukan, menerapkan praktik keamanan terbaik untuk memberikan hak istimewa paling sedikit, atau menonaktifkan fitur yang tidak perlu dalam file konfigurasi.

sistem informasi keamanan dan manajemen acara (SIEM)

Alat dan layanan yang menggabungkan sistem manajemen informasi keamanan (SIM) dan manajemen acara keamanan (SEM). Sistem SIEM mengumpulkan, memantau, dan menganalisis data dari server, jaringan, perangkat, dan sumber lain untuk mendeteksi ancaman dan pelanggaran keamanan, dan untuk menghasilkan peringatan.

otomatisasi respons keamanan

Tindakan yang telah ditentukan dan diprogram yang dirancang untuk secara otomatis merespons atau memulihkan peristiwa keamanan. Otomatisasi ini berfungsi sebagai kontrol keamanan detektif atau responsif yang membantu Anda menerapkan praktik terbaik AWS keamanan. Contoh tindakan respons otomatis termasuk memodifikasi grup keamanan VPC, menambal instans Amazon EC2, atau memutar kredensil.

enkripsi sisi server

Enkripsi data di tujuannya, oleh Layanan AWS yang menerimanya.

kebijakan kontrol layanan (SCP)

Kebijakan yang menyediakan kontrol terpusat atas izin untuk semua akun di organisasi. AWS Organizations SCP menentukan pagar pembatas atau menetapkan batasan pada tindakan yang dapat didelegasikan oleh administrator kepada pengguna atau peran. Anda dapat menggunakan SCP sebagai daftar izin atau daftar penolakan, untuk menentukan layanan atau tindakan mana yang diizinkan atau dilarang. Untuk informasi selengkapnya, lihat Kebijakan kontrol layanan dalam AWS Organizations dokumentasi.

titik akhir layanan

URL titik masuk untuk file Layanan AWS. Anda dapat menggunakan endpoint untuk terhubung secara terprogram ke layanan target. Untuk informasi selengkapnya, lihat Layanan AWS titik akhir di Referensi Umum AWS.

perjanjian tingkat layanan (SLA)

Perjanjian yang menjelaskan apa yang dijanjikan tim TI untuk diberikan kepada pelanggan mereka, seperti waktu kerja dan kinerja layanan.

indikator tingkat layanan (SLI)

Pengukuran aspek kinerja layanan, seperti tingkat kesalahan, ketersediaan, atau throughputnya.

tujuan tingkat layanan (SLO)

Metrik target yang mewakili kesehatan layanan, yang diukur dengan indikator tingkat layanan.

model tanggung jawab bersama

Model yang menjelaskan tanggung jawab yang Anda bagikan AWS untuk keamanan dan kepatuhan cloud. AWS bertanggung jawab atas keamanan cloud, sedangkan Anda bertanggung jawab atas keamanan di cloud. Untuk informasi selengkapnya, lihat Model tanggung jawab bersama.

SIEM

Lihat informasi keamanan dan sistem manajemen acara.

titik kegagalan tunggal (SPOF)

Kegagalan dalam satu komponen penting dari aplikasi yang dapat mengganggu sistem.

SLA

Lihat perjanjian tingkat layanan.

SLI

Lihat indikator tingkat layanan.

SLO

Lihat tujuan tingkat layanan.

split-and-seed model

Pola untuk menskalakan dan mempercepat proyek modernisasi. Ketika fitur baru dan rilis produk didefinisikan, tim inti berpisah untuk membuat tim produk baru. Ini membantu meningkatkan kemampuan dan layanan organisasi Anda, meningkatkan produktivitas pengembang, dan

mendukung inovasi yang cepat. Untuk informasi lebih lanjut, lihat Pendekatan bertahap untuk memodernisasi aplikasi di. AWS Cloud

SPOF

Lihat satu titik kegagalan.

skema bintang

Struktur organisasi database yang menggunakan satu tabel fakta besar untuk menyimpan data transaksional atau terukur dan menggunakan satu atau lebih tabel dimensi yang lebih kecil untuk menyimpan atribut data. Struktur ini dirancang untuk digunakan dalam gudang data atau untuk tujuan intelijen bisnis.

pola ara pencekik

Pendekatan untuk memodernisasi sistem monolitik dengan menulis ulang secara bertahap dan mengganti fungsionalitas sistem sampai sistem warisan dapat dinonaktifkan. Pola ini menggunakan analogi pohon ara yang tumbuh menjadi pohon yang sudah mapan dan akhirnya mengatasi dan menggantikan inangnya. Pola ini diperkenalkan oleh Martin Fowler sebagai cara untuk mengelola risiko saat menulis ulang sistem monolitik. Untuk contoh cara menerapkan pola ini, lihat Memodernisasi layanan web Microsoft ASP.NET (ASMX) lama secara bertahap menggunakan container dan Amazon API Gateway.

subnet

Rentang alamat IP dalam VPC Anda. Subnet harus berada di Availability Zone tunggal.

kontrol pengawasan dan akuisisi data (SCADA)

Di bidang manufaktur, sistem yang menggunakan perangkat keras dan perangkat lunak untuk memantau aset fisik dan operasi produksi.

enkripsi simetris

Algoritma enkripsi yang menggunakan kunci yang sama untuk mengenkripsi dan mendekripsi data.

pengujian sintetis

Menguji sistem dengan cara yang mensimulasikan interaksi pengguna untuk mendeteksi potensi masalah atau untuk memantau kinerja. Anda dapat menggunakan Amazon CloudWatch Synthetics untuk membuat tes ini.

T

tag

Pasangan nilai kunci yang bertindak sebagai metadata untuk mengatur sumber daya Anda. AWS Tanda dapat membantu Anda mengelola, mengidentifikasi, mengatur, dan memfilter sumber daya. Untuk informasi selengkapnya, lihat Menandai AWS sumber daya Anda.

variabel target

Nilai yang Anda coba prediksi dalam ML yang diawasi. Ini juga disebut sebagai variabel hasil. Misalnya, dalam pengaturan manufaktur, variabel target bisa menjadi cacat produk.

daftar tugas

Alat yang digunakan untuk melacak kemajuan melalui runbook. Daftar tugas berisi ikhtisar runbook dan daftar tugas umum yang harus diselesaikan. Untuk setiap tugas umum, itu termasuk perkiraan jumlah waktu yang dibutuhkan, pemilik, dan kemajuan.

lingkungan uji

Lihat lingkungan.

pelatihan

Untuk menyediakan data bagi model ML Anda untuk dipelajari. Data pelatihan harus berisi jawaban yang benar. Algoritma pembelajaran menemukan pola dalam data pelatihan yang memetakan atribut data input ke target (jawaban yang ingin Anda prediksi). Ini menghasilkan model ML yang menangkap pola-pola ini. Anda kemudian dapat menggunakan model ML untuk membuat prediksi pada data baru yang Anda tidak tahu targetnya.

gerbang transit

Hub transit jaringan yang dapat Anda gunakan untuk menghubungkan VPC dan jaringan lokal Anda. Untuk informasi selengkapnya, lihat Apa itu gateway transit dalam AWS Transit Gateway dokumentasi.

alur kerja berbasis batang

Pendekatan di mana pengembang membangun dan menguji fitur secara lokal di cabang fitur dan kemudian menggabungkan perubahan tersebut ke cabang utama. Cabang utama kemudian dibangun untuk pengembangan, praproduksi, dan lingkungan produksi, secara berurutan.

akses tepercaya

Memberikan izin ke layanan yang Anda tentukan untuk melakukan tugas di organisasi Anda di dalam AWS Organizations dan di akunnya atas nama Anda. Layanan tepercaya menciptakan peran terkait layanan di setiap akun, ketika peran itu diperlukan, untuk melakukan tugas manajemen untuk Anda. Untuk informasi selengkapnya, lihat Menggunakan AWS Organizations dengan AWS layanan lain dalam AWS Organizations dokumentasi.

penyetelan

Untuk mengubah aspek proses pelatihan Anda untuk meningkatkan akurasi model ML. Misalnya, Anda dapat melatih model ML dengan membuat set pelabelan, menambahkan label, dan kemudian mengulangi langkah-langkah ini beberapa kali di bawah pengaturan yang berbeda untuk mengoptimalkan model.

tim dua pizza

Sebuah DevOps tim kecil yang bisa Anda beri makan dengan dua pizza. Ukuran tim dua pizza memastikan peluang terbaik untuk berkolaborasi dalam pengembangan perangkat lunak.

U

waswas

Sebuah konsep yang mengacu pada informasi yang tidak tepat, tidak lengkap, atau tidak diketahui yang dapat merusak keandalan model ML prediktif. Ada dua jenis ketidakpastian: ketidakpastian epistemik disebabkan oleh data yang terbatas dan tidak lengkap, sedangkan ketidakpastian aleatorik disebabkan oleh kebisingan dan keacakan yang melekat dalam data. Untuk informasi lebih lanjut, lihat panduan Mengukur ketidakpastian dalam sistem pembelajaran mendalam.

tugas yang tidak terdiferensiasi

Juga dikenal sebagai angkat berat, pekerjaan yang diperlukan untuk membuat dan mengoperasikan aplikasi tetapi itu tidak memberikan nilai langsung kepada pengguna akhir atau memberikan keunggulan kompetitif. Contoh tugas yang tidak terdiferensiasi termasuk pengadaan, pemeliharaan, dan perencanaan kapasitas.

lingkungan atas

Lihat lingkungan.

V

menyedot debu

Operasi pemeliharaan database yang melibatkan pembersihan setelah pembaruan tambahan untuk merebut kembali penyimpanan dan meningkatkan kinerja.

kendali versi

Proses dan alat yang melacak perubahan, seperti perubahan kode sumber dalam repositori.

Peering VPC

Koneksi antara dua VPC yang memungkinkan Anda merutekan lalu lintas dengan menggunakan alamat IP pribadi. Untuk informasi selengkapnya, lihat Apa itu peering VPC di dokumentasi VPC Amazon.

kerentanan

Kelemahan perangkat lunak atau perangkat keras yang membahayakan keamanan sistem.

W

cache hangat

Cache buffer yang berisi data terkini dan relevan yang sering diakses. Instance database dapat membaca dari cache buffer, yang lebih cepat daripada membaca dari memori utama atau disk.

data hangat

Data yang jarang diakses. Saat menanyakan jenis data ini, kueri yang cukup lambat biasanya dapat diterima.

fungsi jendela

Fungsi SQL yang melakukan perhitungan pada sekelompok baris yang berhubungan dengan catatan saat ini. Fungsi jendela berguna untuk memproses tugas, seperti menghitung rata-rata bergerak atau mengakses nilai baris berdasarkan posisi relatif dari baris saat ini.

beban kerja

Kumpulan sumber daya dan kode yang memberikan nilai bisnis, seperti aplikasi yang dihadapi pelanggan atau proses backend.

aliran kerja

Grup fungsional dalam proyek migrasi yang bertanggung jawab atas serangkaian tugas tertentu. Setiap alur kerja independen tetapi mendukung alur kerja lain dalam proyek. Misalnya, alur kerja portofolio bertanggung jawab untuk memprioritaskan aplikasi, perencanaan gelombang, dan mengumpulkan metadata migrasi. Alur kerja portofolio mengirimkan aset ini ke alur kerja migrasi, yang kemudian memigrasikan server dan aplikasi.

CACING

Lihat menulis sekali, baca banyak.

WQF

Lihat Kerangka Kualifikasi Beban Kerja AWS.

tulis sekali, baca banyak (WORM)

Model penyimpanan yang menulis data satu kali dan mencegah data dihapus atau dimodifikasi. Pengguna yang berwenang dapat membaca data sebanyak yang diperlukan, tetapi mereka tidak dapat mengubahnya. Infrastruktur penyimpanan data ini dianggap tidak dapat diubah.

Z

eksploitasi zero-day

Serangan, biasanya malware, yang memanfaatkan kerentanan zero-day.

kerentanan zero-day

Cacat atau kerentanan yang tak tanggung-tanggung dalam sistem produksi. Aktor ancaman dapat menggunakan jenis kerentanan ini untuk menyerang sistem. Pengembang sering menyadari kerentanan sebagai akibat dari serangan tersebut.

aplikasi zombie

Aplikasi yang memiliki CPU rata-rata dan penggunaan memori di bawah 5 persen. Dalam proyek migrasi, adalah umum untuk menghentikan aplikasi ini.

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.