Infrastrcuture as Code

Infrastructure Stacks and building Stacks

Forfattere:
Tor Ivar Melling

Eiere:
Forfatterne og Forskningsstiftelsen TISIP

Innhold

1. INFRASTRUCTUR STACK

En infrastructur stack refererer til en samling av både hardware og software-ressurser som sammen gir en komplett tjeneste. I Azure kan dette inkludere Virtual Machines, databasesystemer som Azure SQL, lagringstjenester som Azure Blob Storage, og nettverksressurser. En instans av stacken er en «levende» versjon av infrastrukturen din, vanligvis i hos en skyleverandør som Azure, AWS etc. Når en kjører *terraform apply*, blir stack-instansen opprettet eller oppdatert basert på stack-koden. Som vi allerede kjenner til, i moderne Infrastructure as Code-verktøy kan konfigurering beskrives deklarativt. For eksempel, i stedet for å manuelt installere en webserver, kan du i koden deklarere at en webserver skal eksistere, og verktøyet tar seg av resten.

1.1. Lavnivå Infrastrukturspråk

Lavnivå infrastrukturspråk, som Shell-skript eller selv AWS CloudFormation, gir deg granulær kontroll, men kan være tidkrevende å vedlikeholde og feilsøke. De er nærmere det faktiske APIet eller kommandolinjeinterfacet for tjenestene de interagerer med.

1.2. Høynivå Infrastrukturspråk

Høynivå infrastrukturspråk som Terraform eller Pulumi tilbyr et høyere abstraksjonsnivå, noe som gjør det enklere å beskrive komplekse systemer. De fokuserer mer på selve infrastrukturen enn på hvordan man interagerer med tjenesteleverandørens APIer (application programming interface).

1.3. Strukturering av Stacks

1.3.1. Monolittisk Stack

En monolittisk stack er en enkelt, gigantisk kodebase som beskriver hele infrastrukturen. Dette kan gjøre det vanskelig å forstå, teste og endre infrastrukturen, spesielt i store team.

1.3.2. Mikro-Stack

En mikro-stack tilnærming deler opp en større infrastruktur i mindre, håndterbare biter. For eksempel kan du ha en stack for nettverksressurser, en for databaser og så videre. Dette øker forståelsen og reduserer feilmarginen. Dvs at en i en mikro-stack-tilnærming kan dele opp Terraform-konfigurasjonen i moduler. For eksempel, kan nettverk, databaser og applikasjonslaget leve i separate moduler.

```
module "network" {
  source = "./modules/network"
  ...
}

module "database" {
  source = "./modules/database"
  ...
}
```

Figur 1 - Terraform moduler

«Building Infrastructure Stacks as Code" betyr å beskrive, benytte versjonskontroller og administrere IT-infrastrukturen på samme måte som en ville gjort med programvarekode. Det gir muligheter for raskere utrulling, enklere endringskontroll og større feiltoleranse. Valget mellom

lavnivå- og høynivå-språk, samt hvordan man strukturerer sin stack, kan ha betydelige implikasjoner for både utviklingshastighet og driftsstabilitet.

1.4. Typiske Bruksområder

Et miljø en samling av operasjonelt relaterte infrastrukturressurser. Det vil si at ressursene i et miljø støtter en bestemt aktivitet, som testing eller kjøring av et system. Det er ofte flere slike miljøer, og hver av dem kjører en instans av det samme systemet.

- Leveringsmiljøer: Støtter en progresjonsprosess for programvareutgivelser. Koden flyttes gjennom ulike miljøer (utvikling, testing, staging osv.) før den endelig går i produksjon.
- Flere Produksjonsmiljøer: Dette gjøres for å oppnå feiltoleranse, skalering og segregasjon. For eksempel, hvis ett miljø feiler, kan de andre fortsatt tilby tjenesten.

Konsistens mellom miljøene er kritisk for å unngå inkonsistent atferd og feil som kan oppstå i overgangen mellom dem. Man kan imidlertid ha spesifikke forskjeller, som størrelsen på miljøene eller tilgangsrettighetene for ulike personer.

1.5. Uheldige mønstre og anbefalt mønster

- Flere-Miljøer-stack: Her håndteres infrastrukturen for flere miljøer i en enkelt stack. Dette kan skape problemer fordi en feil i en del av koden kan påvirke alle miljøene.
- Kopier-og-Lim-Inn miljøer: Her har hvert miljø sin egen kildekodeprosjekt, noe som kan føre til inkonsistens og vedlikeholdsutfordringer over tid.
- Gjenbrukbar stack: Dette er det anbefalte mønsteret hvor man har et enkelt prosjekt som kan brukes til å opprette flere instanser av en stabel. Dette øker konsistens og reduserer feilrisikoen.

For å implementere et gjenbrukbart stack-mønster brukes IaC verktøyet hver gang en vil lage eller oppdatere en instans av stabelen. Med Terraform kan en for eksempel spesifisere forskjellige .tfstate-filer eller forskjellige Terraform Workspace for hver forekomst.

Når den samme infrastrukturkoden blir brukt i flere miljøer, gir det større tillit til at koden vil fungere som den skal i alle andre miljøer.

1.6. Hvorfor Modulbasert Infrastruktur?

Som allerede nevnt i Micro-stakcs, men også for å oppsummere, er modulbasert infrastruktur en praksis der man deler opp hele infrastrukturen i mindre, gjenbrukbare moduler. Dette kan være:

- Nettverksmoduler
- Database-moduler
- Applikasjonsservere
- Load balancers, etc.

Fordelene med en modulær tilnærming inkluderer:

- Gjenbruk: Enkelt å gjenbruke kode for liknende ressurser.
- Isolasjon: Enklere å teste og validere enkeltmoduler.
- Samarbeid: Team kan jobbe parallelt på forskjellige moduler.

2. SIKKERHET OG IAC

Sikkerhet er en kritisk komponent i alle aspekter av systemutvikling og drift, og Infrastructure as Code (IaC) er intet unntak. Når en bruker Terraform for å administrere Azure-ressurser, må flere bestemte sikkerhetshensyn tas i betraktning.

Sensitive Data og Secrets er noe som ofte dukker opp i utvikling og IaC, og en bør for all del unngå hardkoding av sensitive data i Terraform-konfigurasjonsfiler. Dette kan være informasjon som API-nøkler, passord og andre «secrets». I stedet for hardkoding, bruk inputvariabler og krypter disse variablene der det er mulig.

Terraform Vault, fra HashiCorp, selskapet bak Terraform, tilbyr også et produkt kalt Vault for sikker lagring av secrets. Vault kan integreres med Terraform for å hente secrets under kjøring. Når en jobber med Azure, er Azure Key Vault en utmerket tjeneste for å lagre sensitive data og secrets. Ved bruk av rollebasert tilgangsstyring (RBAC), kan en i Azure hjelpe til med å begrense hvilke ressurser en bruker eller tjeneste har tilgang til. En må alltid sørge for å begrense tilgang til bare det som er nødvendig for den spesifikke operasjonen.

Terraform lagrer tilstandsinformasjon som kan inneholde sensitive data. Denne tilstanden må også beskyttes. Dette er filer som terraform.tfstate og terraform.tfstate.backup. Azure tilbyr Azure Storage Service Encryption for å beskytte denne dataen. Om en har sensitivt innhold i disse filene og pusher koden til et åpent github-repository, får en gjerne varsel fra GitGuardian på e-post: *GitGuardian has detected the following Microsoft Azure Storage Account Key exposed within your GitHub account*

Infrastrukturkode bør under samme strenge kodegjennomgangsprosesser som applikasjonskode for å sikre at det ikke er noen svakheter eller sensitiv informasjon som kan komme på avveie. Verktøy som HashiCorp Sentinel eller Azure Policy kan brukes for å definere policyer for infrastruktur som kode, noe som gir et ekstra lag av sikkerhet.

Ved å følge kjente gode praksiser kan en bygge en sikker, robust og pålitelig infrastruktur med Terraform og Azure.