Terraform (코노멘트)

#SW마에스트로

알고 싶은 이유

- 선우
  - 인프라를 코드로 관리하기 때문에 실수를 줄이고, 자동화를 할 때 효율적일 것 같다.
  - 앞으로 배포를 할 때 편할 것 같다.
  - 오픈소스 컨트리뷰션에서 테라폼 네이버 해보았는데, 테라폼을 알면 더 컨트리뷰션할 기회가 생길 것이라 생각
- 채연
  - 혼자 공부할 때는 이것저것 찍먹하기 어려움
  - 소마에 온 이상 인프라도 찍먹!

궁금한 점

- 테라폼은 라이센스 변경으로 기업들에게 비싸진걸로 아는데 Pulumi를 쓰지 않는 이유가 있나요?
  - 테라폼 라이센스 변경 (작년 말쯤)
    - BSL 라이센스 변경 (오픈소스 라이센스 X)
      - 코드는 공개 / 상업적으로 사용할 때 하시코프와 상의
      - 테라폼 을 랩핑해가지고 상업적으로 제공할 수 없음
    - 엔드유저는 직접적 영향이 없음
    - 다만, 간접적 영향 발생
      - 고객들이 서드파티 서비스 이용하는데 문제가 생김
    - 하시코프에서 만든 도구 중 가장 유명한 도구
      - 근데 이걸 가지고 돈을 못벌어요
      - Terraform Enterprise
        - 실패
      - Terraform Cloud
        - 잘 만들었지만, 가격정책이 이상해서 계속 실패
        - 작년 6월 엄청난 가격 정책 변화
          - 실패
          - 기존
            - 인원 수 / 프로젝트 수 / apply 수 팩터로 가격 계산
          - 변경
            - 코드로 관리하는 리소스 수 기반 계산

        - 서드파티 회사들에서 Terraform Cloud 대체제를 서비스하기 시작
            - Spacelift
            - Scalr
            - Env0
        - OpenTufu

사전 준비 (Pre-requisites)

- 테라폼 설치
- AWS 개인 계정
- AWS CLI
- AWS CLI 인증 설정

인프라 관리 도구

IaC (Infrastructure as Code)

> 인프라를 코드로 관리할 수 있게 해주는 도구
>
> 클라우드 리소스(AWS IAM User, S3 Bucket, GitHub Organization, GitHub Repository)를 프로비저닝하는 것에 초점 (네트워크, 가상머신, 데이터베이스 등)

- **Terraform**
    - HashiCorp 에서 만듦
        - 인프라 분야에서 굉장히 유명한 회사
        - 개발 겁나 잘함 / 영업 겁나 못함
            - 수익이 안나요 / 적자 회사
            - 3년 전인가 나스닥 상장
            - 삼토막 났음
            - **2024년 4월에 IBM이 하시코프 인수 발표**
        - Provider-agnostic
        - Vagrant
        - Packer
        - Terraform
        - Consul
        - Nomad
        - Vault
        - Boundary

    - **HCL (HashiCorp Configuration Language)** 설정 언어
    - Provider-agnostic
        - 하나의 테라폼 코드로 AWS EC2 / GCP 서버 / Azure 서버 를 다 만들수 있냐?
        - 가능은 한데 비효율적이다.
            - EC2의 기능과 GCP 서버의 기능은 다를 수 밖에 없다.
- Pulumi
    - 기존의 프로그래밍 언어를 사용
        - Node.js / Python / Go
- AWS CloudFormation
- **CDK (Cloud Development Kit)**
    - 기존의 프로그래밍 언어 사용
        - Node.js / Python / GO 등
    - **CDK for Terraform**
        - 프로그래밍 언어로 테라폼 쓰듯이 인프라 정의하고 나면
        - 산출물이 테라폼 코드가 되요
        - 동적 테라폼 프로그래밍
    - CDK for AWS
    - CDK for Kubernetes
- 가능은 함
    - Ansible
- IaC를 하는 장점
    - 인프라가 코드로 관리된다는 것
        - GitOps
            - 버전 관리
            - 코드 리뷰
            - 코드 기반의 문서화
            - Lint / Test
    - 반복이 쉽다
- IaC 단점
    - 모든 클라우드 리소스 혹은 옵션을 100% 커버할 수 없음
        - Why?
            - 라이브러리(프로바이더) 코드가 서비스 프로바이더의 릴리즈를 같은 속도로 따라가지 못함
                - 메이저한 서비스들은 릴리즈 속도가 빨라요
                - 마이너한 서비스들은 릴리즈 속도가 느리고 안될 때도 있다
                    - 그러면 어떻게 해야 하나?
                        - 그 프로바이더(오픈소스) 포킹해가지고 직접 수정하면 됨
                            - 이것도 안될 때가 있어요

        - 프로바이더가 API를 열어주지 않는 경우
            - 웹 콘솔에서만 작업이 가능한 기능
            - CLI에만 작업이 가능한 경우
    - 1회성 작업인 경우 / 한 번 하고 나면 오랫동안 안건드리는 경우

Image Builder

> 특정 플랫폼(AWS EC2, Docker, VMWare, VirtualBox, Parallels, GCP 서버)의 실행 이미지를 빌드하는 것

- **Packer**
- AWS Image Builder

CM Tools (Configuration Management)

> 설정 관리 / 형상 관리
>
> IaC 에서 서버가 프로비저닝이 되면, 그 이후에 운영체제 설정 / 패키지 설치 및 설정 / 어플리케이션 배포 등

- **Ansible**
    - vs Shell Script
        - 명령형 / 순차적 실행 (Imperative)
        - 멱등성을 보장하지 않아요 (기본)
    - 최종 상태를 정의 (선언형 ➜ Declarative)
        - 멱등성을 보장 (기본)
            - 잘 짜야 보장된다
    - Agent 설치가 불필요
        - SSH 기반
- Puppet
    - Agent 기반
- Chef
    - Agent 기반
- SaltStack
    - Agent 기반

테라폼 (Terraform)

- Documentation | Terraform | HashiCorp Developer
- Terraform Registry

목표

- VPC 네트워크를 코드로 다뤄보기
- 테라폼의 기본문법을 핸즈온

핵심 개념 (Core Concepts)

프로바이더 (Provider)

- 라이브러리
- 어떤 클라우드 서비스를 코드로 관리할거냐?
- 예시
  - AWS
  - GCP
  - Azure
  - GitHub
  - Datadog
  - Grafana
  - Kubernets

모듈 (Module)

- 객체지향의 클래스?
- 리소스 / 데이터 소스 / 로컬 변수 / 입력 변수 / 출력변수를 묶어서 추상화한거

레지스트리 (Registry)

- 프로바이더나 모듈을 공유하기 위한 목적
- 공개 레지스트리 (Public Registry)
  - Terraform Registry
- **비공개 레지스트리 (Private Registry)**
  - GitHub
  - GitLab
  - AWS S3
  - **Terraform Cloud / Enterprise**
  - **서드파티**

워크스페이스 (Workspace)

- 코드로 인프라를 관리할 때 만들어지는 테라폼 상태 단위

테라폼 상태 (Terraform State)

- 코드로 인프라를 찍어내겠죠? 그때 코드가 어떤 인프라 리소스를 관리하고 있는지를 기록해두는 것
- `terraform.tfstate` 라는 파일로 만들어짐
-

테라폼 상태 저장소 (Terraform State Backend / Terraform State Storage)

- 로컬 상태 저장소 ➔ 테스트 / 혼자 관리 목적이 아니면 권장하지 않음
  - `terraform.tfstate` 라는 파일로
  - 버전 동기화가 안될 수 있음
  - 민감한 데이터가 포함될 수 있음
- 원격 상태 저장소 (Remote State Backend)
  - 기능
    - 네트워크 저장소
    - 잠금 (Locking)
  - **AWS S3 Bucket + DynamoDB Table**
    - Terraform Cloud / Terraform Enterprise 가 나오기 전에 메이저로 자리 잡음
    - 닭과 계란 문제
  - **Terraform Cloud / Terraform Enterprise**
  - Kubernetes
  - HashiCorp Consul
  - **서드파티**
    - Scalr / Env0 등

리소스 (Resource)

- 인프라에 프로비저닝하고자 하는 실질적인 요소/자원
- 보통 API 엔드포인트 1개 = 테라폼 리소스 1개
- `resource_type.resource_name.resoure_property`
  - `aws_vpc.main.id`

데이터 소스 (Data Source)

- 이미 있는 데이터를 조회하여 활용하기 위한 목적
- `data` 블록 정의
- `data.${data-type}.${data-name}.${data-attriute}` 로 참조

로컬 변수 (Local Variables)

- `locals` 블록을 정의
  - 블록의 이름이 없어요
  - 여러 지역 변수를 정의할 수 있어요
- 참조할 때는 `local.{variable_name}`
- 중간 계산 결과 값을 저장하여 활용하기 위한 목적으로 사용
- 정의 순서 상관 없음

입력 변수 (Input Variables)

- `variable` 블록으로 정의
- 참조 시에는 `var.{variable_name}` 으로 참조

- `type`으로 데이터 타입 명시
  - 명시하지 않으면 `any` 라는 특수 타입
- 변수 넘기는 방법
  - `-var k=v`
  - `-var-file asdf.tfvars`
    - 보통 변수 파일의 확장자는 `.tfvars`
      - `dev.tfvars`
      - `prod.tfvars`
      - `staging.tfvars`
  - 환경변수 `TF_VAR_variable_name`
  - `terraform.tfvars` / `.auto.tfvars`
    - 자동으로 읽음
  - 우선순위에 대해 이해해야 함

출력 변수 (Output Variables)

- 테라폼 워크스페이스의 결과값을 정의
- 자동화 과정 등에서 통합(Integration) 목적으로 활용 가능

디렉토리 구성

기본 워크플로우

- init
- (plan)
- apply