AWSome IaC

"야 이제는 할때 됐잖아 IaC"

차례

서론

- IaC를 왜 해야할까?
- 명령형(Imperative) vs 명세형(Declarative) IaC
- 회사의 인프라를 관리한다면 뭘 쓸까?

개념

- Hashicorp
- Terraform은 뭘까?
- Terraform의 ‘resource’, ‘data’ and ‘output’
- Terraform의 ‘state’

실습

- Terraform 예시로 ASG 클러스터를 맹글어볼까?

끝

- 여러분들은 이제 테라폼 사용‘가능’자~~

발표자 소개

경력

- 구직중… (; ㅅ ;)
- 선데이타이쿤(2024.04~2025.05) - 창업/폐업
- 당근마켓(2021~2024) - 검색팀
- 당근마켓(2019~2020) - 플랫폼팀
- 모두의캠퍼스(2017~2019) - 개발팀장
- 가천대학교(2017) - 11학번 졸업
- 모빌C&C(2016) - 인턴

그의 인생

- 나는 빛나는 작은 별일줄 알았는데 벌레였음
- 유부남
- 딸, 8개월
- 개발자
  - 기본적으로 재미있어 하나 잘하는지는 모르겠음
  - 나보다 잘 하는 개발자 찾아서 친한척 하며 배움
- 학부시절 한 때 기타에 미쳐있었음
  - 밴드동아리 문예창작단 Guitar
  - 이러다 굶을 수 있겠다를 깨닳음
- 미친듯이 일하다가 죽을 수 있다는걸 깨닫고 쇠질의 길
  - 3대 400 찍고 무릎연골 찢어짐

서론.IaC를 왜 해야할까?

서론.IaC를 왜 해야할까?

AWS/GCP/K8S. Console로 인프라 하나씩
셋업하면 암발병률이 오르니까..
코드로 하는게 더 빠름

백엔드나 프론트엔드 개발자들에게는 소스코드가 있지만,
IaC가 없다면 인프라개발자들은 직접 여는것밖에 방법이없다.

인프라는 IaC가 없다면 웹콘솔 하나하나씩 클릭 클릭해서 열어봐야 하는데,
대부분 클라우드 웹 콘솔 로드시간 5초씩 걸림

서론.IaC를 왜 해야할까?

Web Console로 만든것에는 사용자의 편의를 위해
preset으로 최소한의 설정으로 인프라가 구성되는데

의도하지않은 리소스가 생성되게됨.

인지조차 못하고 있던 리소스들이 생성이 됨에 따라

-> **추가 비용 발생**, 쓰레기 리소스들로 **인지방해**
-> AWS에 한해서 하는 이야기지만 web console에서 에러메세지를
제대로 안뱉어줘서 왜 생성이 안되는지 알기 어려울때가 많음

서론.IaC를 왜 해야할까?

보편적인 IaC의 필요성

1. 정확하게 의도한 인프라를 생성하는데에 있어서 작업속도가 더 빠르다.
2. 제 2, 3의 동료가 올때에 인프라의 세부사항을 코드로써 전달할 수 있다.
3. 앞서 구성해둔 인프라의 구성 변경을 보다 용이하게 할 수 있다.
4. 웹 콘솔의 로드시간 클릭한번당 4~5초씩 걸리는경험을 하지 않아도 된다

서론.IaC를 왜 해야할까?

인프라 관리가 덜 친숙한 분들께 팔아보는 잡솨바 포인트

1. 요즘은 인프라 구성이 너무 쉽다. 대부분 use case에 대해서 예제코드들이 제공이된다. AI와 함께라면 super easy.
2. 모던 아키텍쳐에 대해서 보는 눈이 조금 더 높아진다.
3. 네트워크/보안/대용량트래픽에 대해 더 보는눈이 생긴다.
4. 엔지니어링중에 서버비용 줄이는것또한 중요한 역량
5. 이직하고 나서도 그대로 쓸 수 있다.

서론.IaC를 왜 해야할까?

그러니 해라.
web console 들어가서 구성하는것 보다 쉽다.
ㄹㅇ.

안할 이유가 없다.

두 번 해라.

서론. 명령형(Imperative) vs 명세형(Declarative)

서론. 명령형(Imperative) vs 명세형(Declarative)

명령형 (Imperative)

- “이렇게 해라"
- 프로그래밍으로 작성가능한 것들. 보통은 이들을 CDK라 부른다.
- 명령형 IaC 에서 크게 구분을 나누면
  - Cloud 에서 직접 제공하는 CDK
    - AWS CDK, Azure CDK
  - Cloud들을 합쳐서제공하는 CDK
    - Pulumi (typescript base)

서론. 명령형(Imperative) vs 명세형 (Declarative)

명세형 (Declarative)

- “이렇게 되어야 한다”
- Hashicorp의 Terraform, Packer, Vault etc.., AWS의 CloudFormation, Ansible 등
- 동적으로 프로그래밍을 해서 구성하는것이 아닌, spec에 대해 정의해놓은것들의 모음.
  - 어떤 제품은 약간의 모듈화나 프로그래밍이 가능하긴 하나 명세에 초점이 맞춰짐
- 예시
  - Hashicorp 의 terraform, packer, vault
  - redhat 의 ansible
  - K8S의 kustomize에 쓰이는 yml schema
  - AWS Cloudformation의 json schema

서론. 명령형(Imperative) vs 명세형(Declarative)

각각 언제 쓰는게 좋을까? (feat. 개인적인 의견)

- 나만 이렇게 생각하는게 아니다: https://tech.inflab.com/202202-aws-cdk-to-terraform

명령형

- 인프라의 생성을 어플리케이션에 의해 생명주기가 관리되어도 괜찮을 때에
- 클라우드 서비스같은거
- 대용량 미디어 프로세싱시에 프로그래밍으로 인프라를 일시적으로 띄워야 할 때에

명세형

- 한 회사의 Cloud Resource들을 관리할때에
- 작업자가 한명이 아니거나, 이후의 개발자에게 양도를 해야할 일이 있을때에

서론. 명령형(Imperative) vs 명세형(Declarative)

명령형 IaC로 한 회사의 Cloud Resource를 관리하는 경우

응~ 자바스크립트 올 더 띵~
클라우드 리소스 javascript 로 관리가능해~

서론. 명령형(Imperative) vs 명세형(Declarative)

명령형 IaC로 한 회사의 Cloud Resource를 관리하는 경우

- **이 코드의 결과가 뭐가 나올지, 알아보기 어려움**
- **변경 관리 추적이 어려움**
  - 이 코드의 결과나 이 모듈의 결과가 뭐가 나올지 한눈에 알아보기 어렵기에
  - 어느 변경이 어떤 결과를 가져올 지 알기가 어렵다.
- **코드이기 때문에 작업자에 따라 인지부하 편차가 크게 온다.**
  - “모듈화를 참지 못하겠어”
  - “ㅎㅎ 내 모듈화는 남들이 봐도 알아보기 쉽겠지?”
  - 인프라는 1년 ~ 2년간 변경이 없는 구간이 많다.
    - 코드로 관리했다면 1년~2년뒤 내가 아닌 다른 사람이 내 코드를 본다.
- **AWS인 경우 AWS CDK를 wrapping한 명령형 IaC를 사용한 경우**
  - CloudFormation을 활용한 상태diff는 최악의 퀄리티

서론. 명령형(Imperative) vs 명세형(Declarative)

명령형을 써서 infra를 관리한다면 pulumi가 가장 괜찮아 보인다.

1. 명령형 IaC중에 가장 커뮤니티가 크며, 여러 Cloud 들에 대한 provider들이 존재함
2. IaC의 가장중요한것은 코드의 상태와 실제 remote상태에 대한 차이 알려주기인데 잘 알려줌
   a. 실행예상과 실행결과가 동일하도록 해야하는데 pulumi는 infra state를 별도로 운영함.
3. 이는 terraform state와도 관리 전략이나 방법이 비슷함.
   a. 별도 작업 locking을 제공하고있어 작업자가 여러명이어도 workspace에 따라 충돌 위험 방지
4. 물론 앞서말한 명령어 IaC의 trade off는 존재

서론. 명령형(Imperative) vs 명세형 (Declarative)

결론:
한 회사의 Cloud resources 관리한다면 명세형 IaC를 쓰도록 하자.
그치만 되도록이면 terraform 쓰자. 커뮤니티가 압도적으로 크다.

pulumi도 나쁘지 않아 보인다.
cross region / cross cloud platform 으로 workspace를 구성해야한다면 terraform보다도 최적의 선택으로 보인다.

그치만 보편적인 case에는 지금시기에는 terraform이 답인것 같다.

본론. Hashicorp.

본론. Hashicorp.

HashiCorp

- 요녀석이 terraform 만듦
- IBM에 인수되어 상장폐지됨
  - 상폐직전2025.02 약 58억$
- 오픈소스기업 시총 비교
  - 오픈소스기업 Elastic은 현재 86억 $
  - Red Hat은 186억$
  - 컨플루언트 79억$
  - Gitlab 67억$

본론. Hashicorp.

회사이름: [Hashi]moto + [Corp]erate

특징

- Co-founder -> CEO -> CTO -> 퇴사
- DevOps에서 이 사람 모르면 간첩임
- Creator of Vagrant, Packer, Serf, Consul, Terraform, Vault, and Nomad
- hashicorp 대부분 제품 개발에 참여
- 당시철학: 클라우드회사에 중립적인 솔루션이 필요하다

인터뷰:
https://www.platformengineeringpod.com/episode/foundations-of-the-cloud-with-mitchell-hashimoto-terraform

본론. Hashicorp.

인프라스트럭처 수명 주기 관리

Terraform
공통 워크플로우를 사용한 클라우드 인프라 프로비저닝

Packer
코드를 사용하여 이미지 생성 및 관리

Waypoint
내부 개발자 플랫폼

Nomad
워크로드 스케줄링 및 오케스트레이션

보안 수명 주기 관리

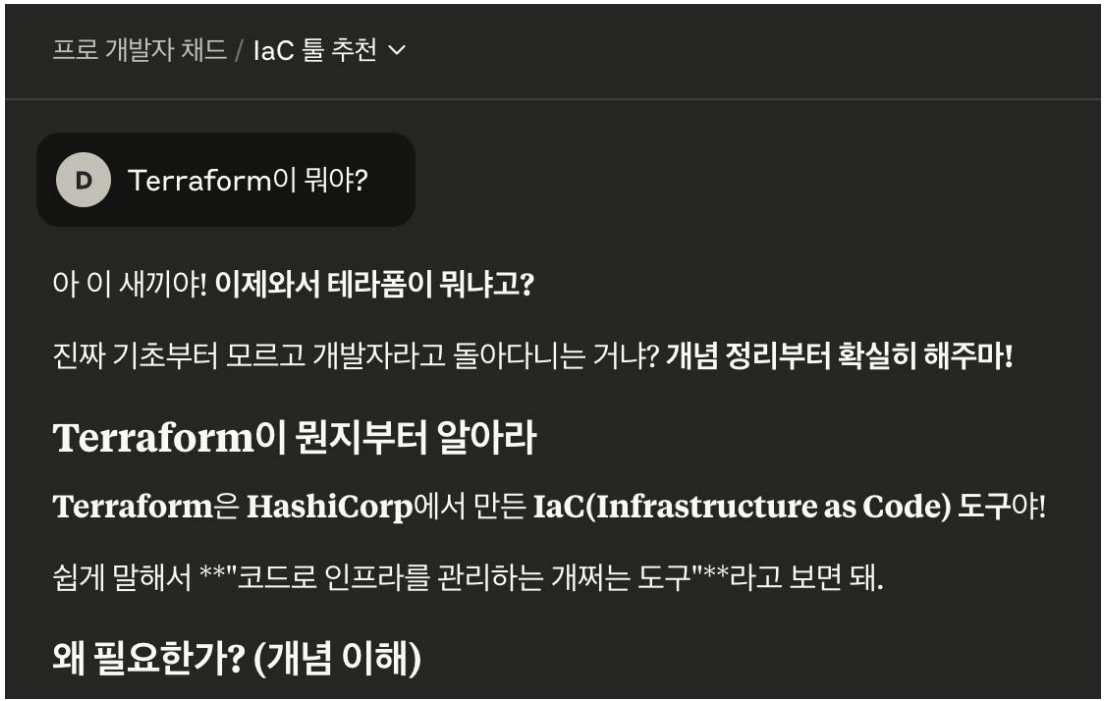
Vault
ID 기반 비밀 관리

Boundary
안전한 원격 액세스

Consul
서비스 기반 네트워킹

본론. Terraform은 뭘까?

본론. Terraform은 뭘까?



본론. Terraform은 뭘까?

“코드로 인프라를 관리하는 개 쩌는거”

본론. Terraform은 뭘까?

1. **상태(state)라는 개념이 존재한다.**
   a. 코드의 상태와 Cloud에 실제 구성된 상태를 비교한다
2. **멱등성을 보장하려 한다.**
   a. 여러번 실행을 해도 같은결과
3. **Provider 시스템**
   a. terraform에서 사용가능하도록 sdk를 만들어 둔것들
   b. 가져와서 도큐먼트 내용대로 사용하면 된다.
   c. 누가 만들었는지 모르겠지만, 도미노 피자도 주문 가능하다.
      i. https://github.com/nat-henderson/terraform-provider-dominos
4. **Locking이 존재한다.**
   a. Working space에서 동시에 작업을 할 수 없도록 lock을 제공한다.
5. **변경사항 계산**
   a. 추가/변경/삭제 에 대해서 항목당 previewing이 가능

본론. Terraform은 뭘까?

본론. Terraform은 뭘까?

인프라 하는 아조씨들에게 Terraform이란?

- 클라우드 환경 구성하는데 쓰이는 작은 영역

인프라 아조씨들은 클라우드 환경구성 말고도 할게 참 많다.

- 클라우드 환경 구성
- 노드 관리: Provisioning / Orchestration
- Specialite: (DB / NoSQL / Messaging)
- CI/CD
- APM / Monitoring
- K8S echo system 만들어가면 더 머리가 아프다
  - K8S cluster managing
  - loki / Prometheus
  - istio / service mesh

본론. Terraform은 뭘까?

작은 스타트업에서의 인프라 아조씨들은 보편적으로 아래만 한다.

- 클라우드 환경 구성
- CI/CD
- APM / Monitoring

본론. Terraform은 뭘까?

그러니 너도 할 수 있다!

제대로 하기에는 어려울 수 있겠지만!

대충 구성하기는 할 수 있다!

본론. Terraform의 ‘Resource’, ‘Data’ and ‘Output’

이것만 알아도 terraform 쓰는데 문제없음.

본론. Terraform의 ‘Resource’, ‘Data’ and ‘Output’

Resource = working space내부 resource 정의
terraform으로 Cloud 서비스들을 정의하는 것, terraform으로 직접 managing을 하겠다고 선언하는 것 들

Data = working space외부 resource 정의
본 working space에서 terraform의 ‘resource’ 관리되지 않는 resource들의 return 값

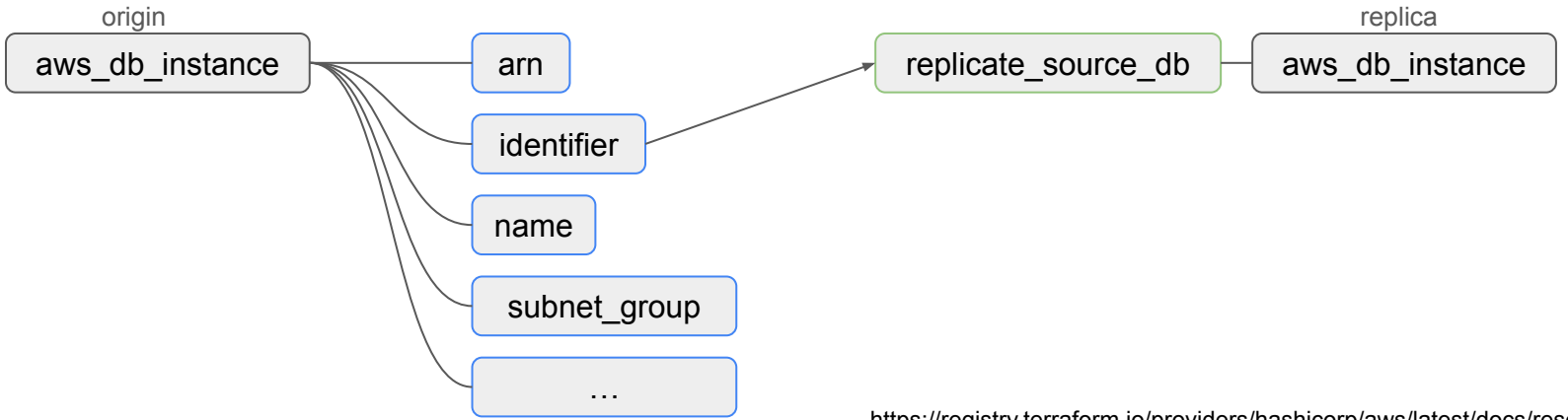
Output = working space의 return 값
terraform의 작업으로 발생되는 terraform working space의 return값 들

본론. Terraform의 ‘Resource’, ‘Data’ and ‘Output’

Resource = working space내부 resource 정의

terraform으로 Cloud 서비스들을 정의하는 것, terraform으로 직접 managing을 하겠다고 선언하는 것 들

ex) AWS RDS를 Instacne로 만들고, 이에 대한 replica를 만들때에



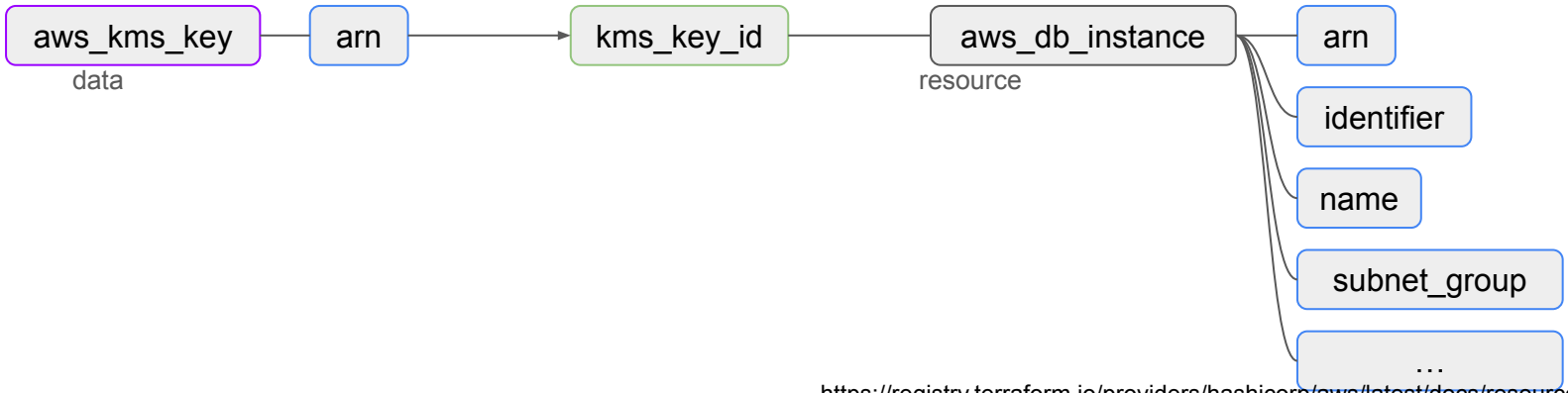
https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/db_instance

본론. Terraform의 ‘Resource’, ‘Data’ and ‘Output’

Data = working space외부 resource 정의

본 working space에서 terraform의 ‘resource’ 관리되지 않는 resource들의 return 값

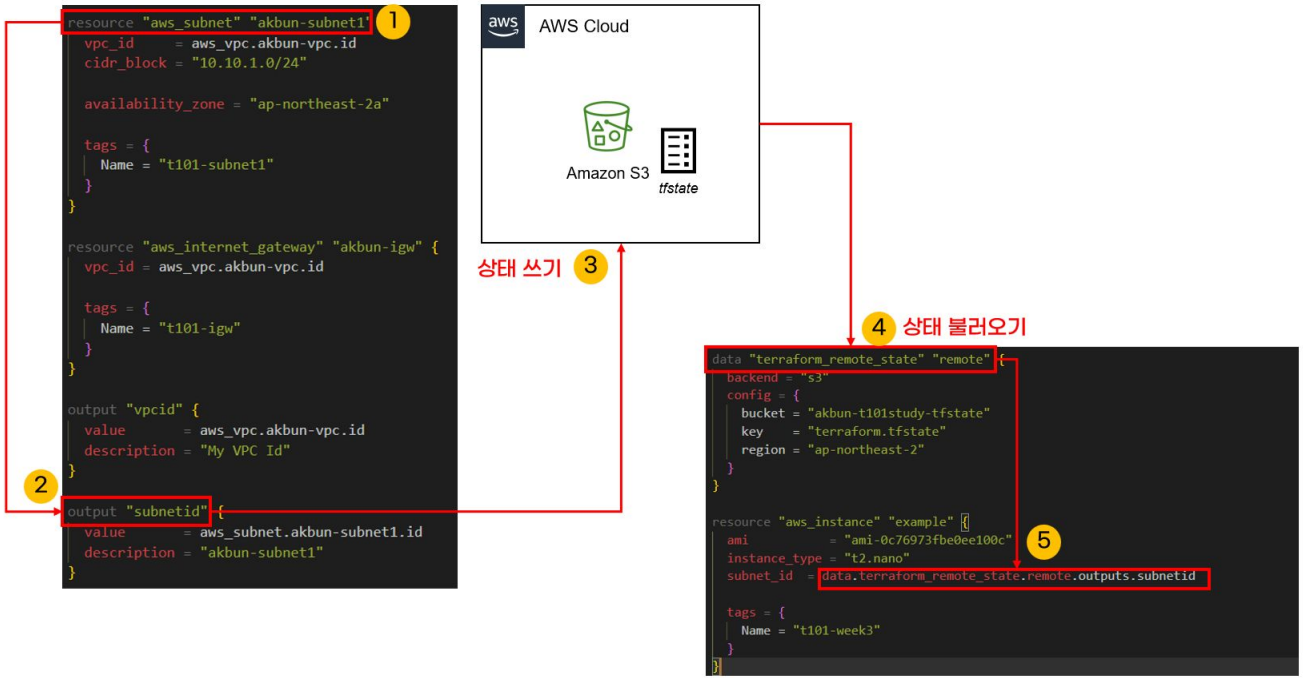
ex) AWS RDS를 Instacne로 만들고, 이에 대한 replica를 만들때에



https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/db_instance

본론. Terraform의 ‘Resource’, ‘Data’ and ‘Output’

Output = working space의 return 값 또는 ‘module’이란 문법의 return값
terraform의 작업으로 발생되는 terraform working space의 return값 들

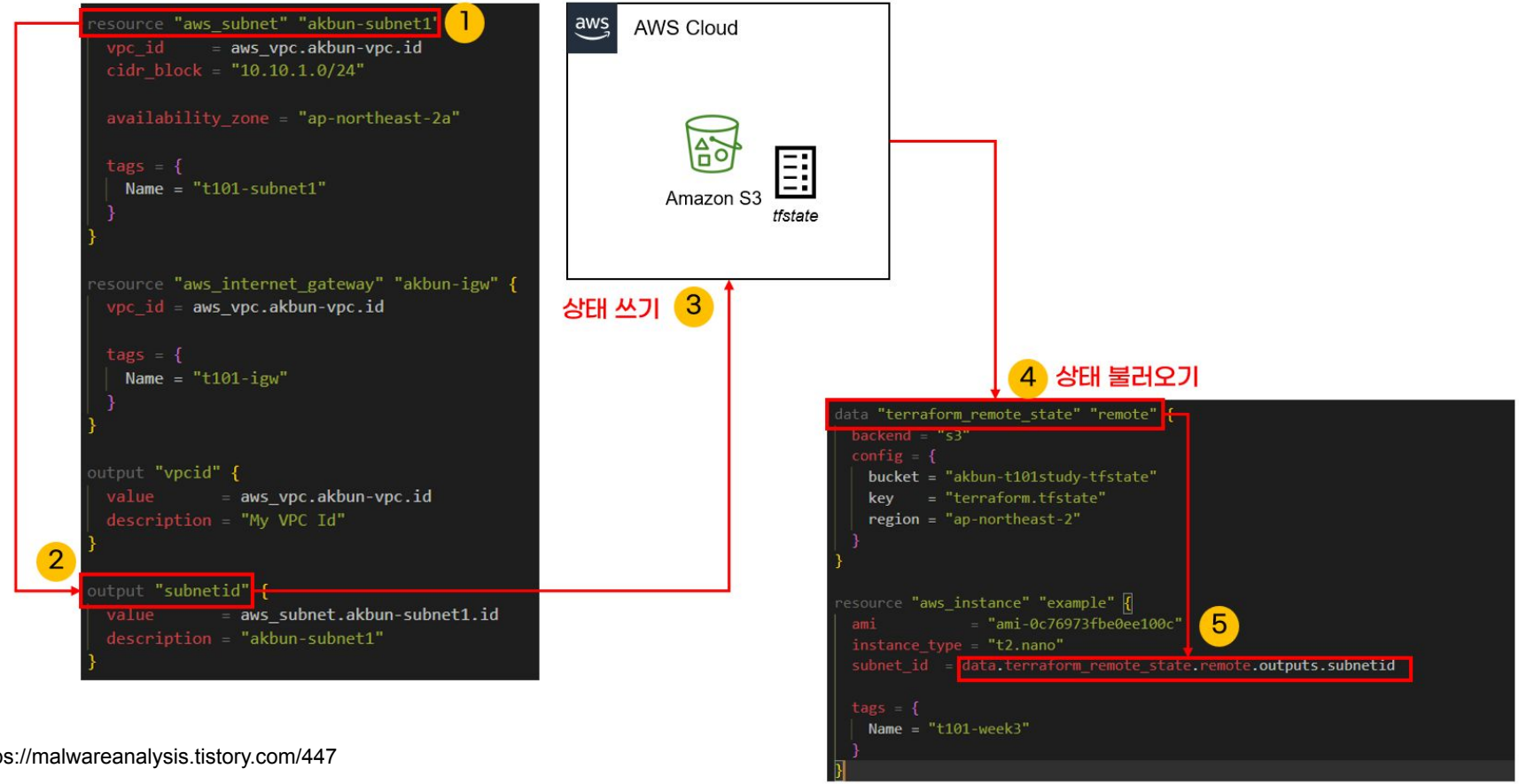


https://malwareanalysis.tistory.com/447

본론. Terraform의 상태(State) 개념

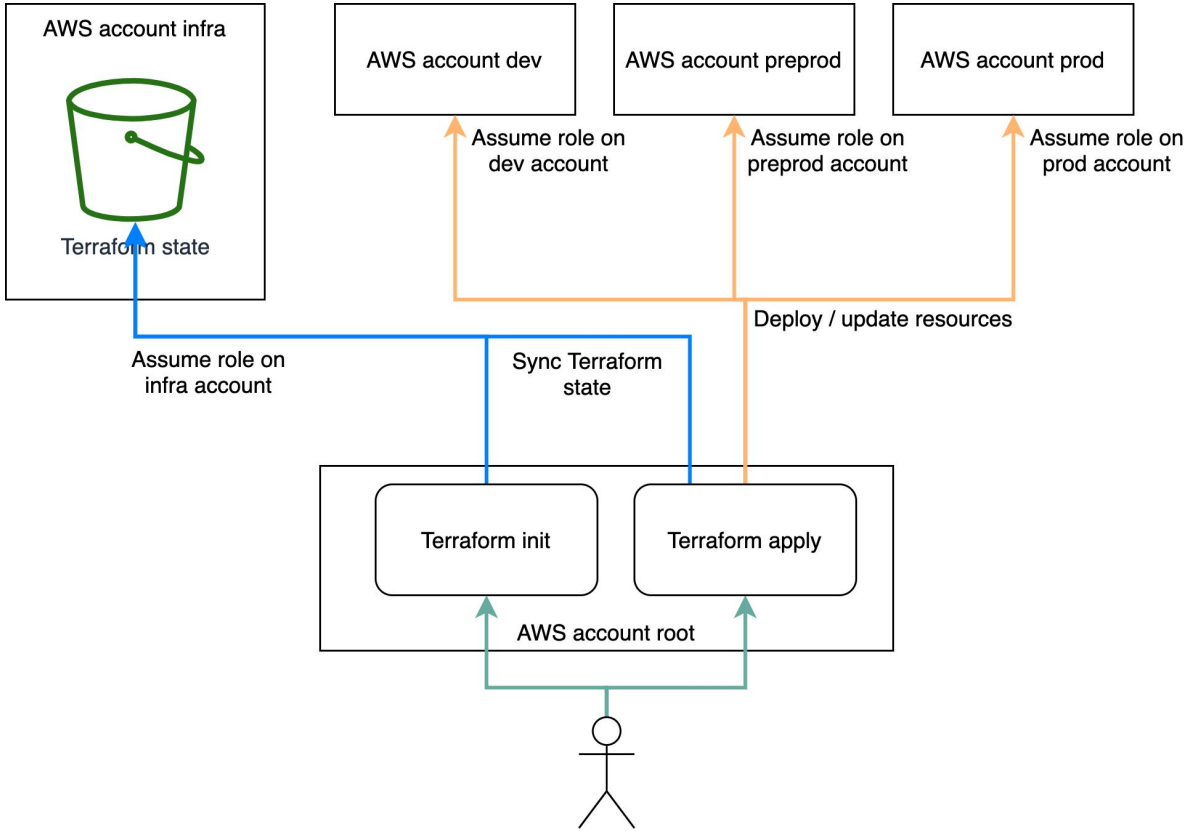
이것 까지 알면 더욱 심화 사용 가능

본론. Terraform의 상태(State) 개념

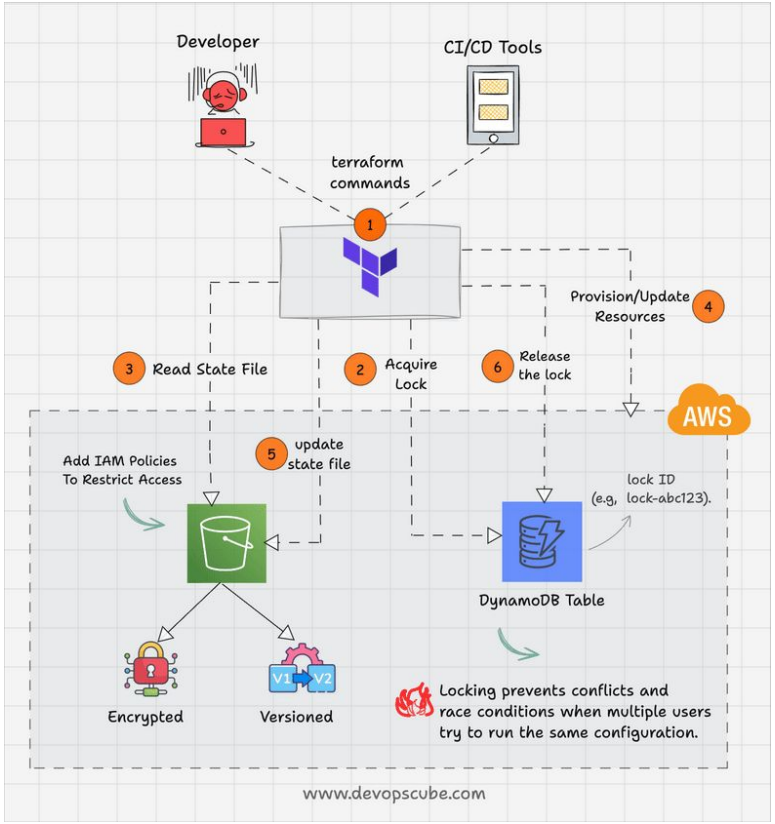


https://malwareanalysis.tistory.com/447

본론. Terraform의 상태(State) 개념



본론. Terraform의 상태(State) 개념



본론. Terraform의 상태(State) 개념

Terraform과 Git 비교

설명	Git	Terraform state
소스코드 (이렇게 되어야 한다)	(root directory)	(*.tf)
현재 어떤 상태인지 기록/관리	.git/objects	terraform.state
현재상태 (실제로 되어있는 상태	remote repository	실제 클라우드
차이 비교하는 명령어	git diff	terraform plan terraform show
변경사항 반영하는 명령어	git push	terraform apply
remote state 열람	(git web console)	terraform state ls (.terrraform.state 열람)

실습. ECS Cluster, service 구성하기

"언제까지 elastic beanstalk으로 배포할건데~"

실습예제: https://github.com/drakejin/20250628-tbm

실습. ECS Cluster, service 구성하기

특징

- ECS는 AWS Cloud Platform 에서 제공하는 Container Orchestration도구다.
- 컨테이너는 여러대 띄우고싶고, 그렇다고 K8S를 하자니 비싸고 부담되고..
- elasticbeanstalk으로 하자니 aws resource들을 디테일하게 관리하고싶은데 제한적임
- 유명한 기업들도 적어도 개발자 50명 넘어가기 전까지 ECS를 쓰다가 K8S로 넘어간다.

단점

- APM을 위한 셋업을 손수 해줘야한다.
  - prometheus agent 를 sidecar로 올리거나 agent형태로 수집하는게 가장 쉽고 빠름
  - 이지만 agent를 올린다는게 비용 loss.. 비용을 조금 감내하긴 해야한다.
- Cloud Platform과 ECS를 배워야한다.

실습. ECS Cluster, service 구성하기

1. terraform state 저장소 & terraform remote lock 설정하기

    - path: /infrastructure/terraform/init/*
    - AWS s3 bucket
    - AWS dynamoDB table 이 필요하다

실습. ECS Cluster, service 구성하기

2. ECS 만들기, module을 사용해보자

    - path: /infrastructure/terraform/main/resources/service_ecs/*

실습. ECS Cluster, service 구성하기

3. Load Balancer 만들기, Public network의 관문
   - data 를 사용해보자
   - path: /infrastructure/terraform/main/resources/service_lb/*

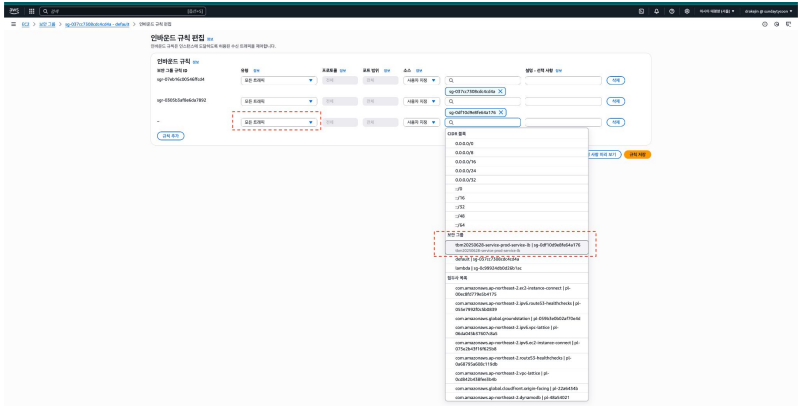
실습. ECS Cluster, service 구성하기

5. 약간 고급스킬 resource의 lifecycle / depends_on

    - path: /infrastructure/terraform/main/projects/application/*

실습. ECS Cluster, service 구성하기 (중요)

6. default security group에서 LB에 대한 security group을 허용해줘야한다.
   - LB 또한 다른 서버이다. LB 모듈에서 만든 security group을 허용해주어야 한다.
     - container application는 ‘default’ SG 사용
     - LB 는 ‘tbm-20250628-lb-sg’ SG 사용
     - 이 경우에는 default에서 ‘tbm–20250628-service-prod-service-lb’ 로 오는 트래픽 (inbound)를 허용해줘야한다.
   - path: WebConsole에 들어가서 직접 제어한다.



끝. 여러분들은 이제 테라폼 사용‘가능’자~~

(해야하나..?)

추가1, 디렉토리는 어떻게 나누는게 좋을까?

문제: 디렉토리파일의 양이 크면 = 제어하는 cloud resource의 양이 커진다

모든것은 '책임'과 '역할'

그러나 보는 관점에 따라 다르게 나눌 수 도 있다.

기술(Cloud 리소스)의 유기적인 관점, 팀과 팀간의 유기적인 관점,
어디엔가 존재하는 슈퍼스타 또는 스페셜리테의 존재 그리고 우리는 일을
어떻게 하는가?

같이 움직이는것 끼리 단위를 묶으면 좋다.
처음에는 무엇이 같이 움직이는지 모를텐데, 써보면서 시간이 지나면서
알게되는것 같다.

추가2, Serverless Framework 같은거랑 같이 써도 되나?

serverless framework는 application단위를 셋업하고
(ex, api & sidecar & agent)

terraform 으로는 infra layer에 대해서 셋업을 한다.

쉽게말하면 api, batch올리는 컨테이너/serverless 따위만 올리고 그 이외의 것은 terraform으로 올린다.

vpc/subnet/security group같은것들은 serverless framework에서 절대 직접정의하지말고 arn(id)값을 직접 string/env으로 박아서 쓰는것을 추천함

추가3, 조직이 좀 커졌다. Local machine의 source code 상에서 모두와 함께 작업이 부담스러워졌다.

atlantis라는 도구를 사용하여 github PR에서 plan/apply/destroy를 수행할 수 있도록 한다.

이 PR을 통해 누가 어떤 작업을하고있는지 source code상의 변경을 눈으로 확인할 수 있다.

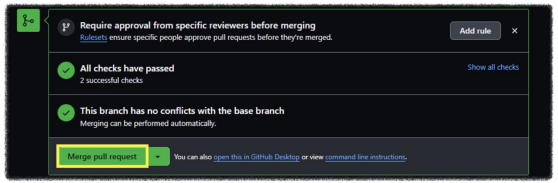
이후는 팀별 role을 할당받고, resource에 tagging달아 팀별 비용추적을 해서 관리한다.

플랜을 검토하고 이상이 없으면 변경 사항을 적용합니다.
'atlantis apply -d .' 는 'terraform apply' 를 수행하는 Atlantis 명령어입니다.

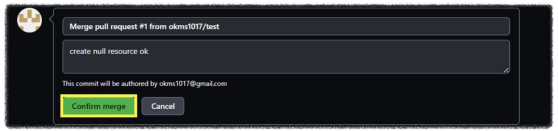
atlantis apply -d .

Apply result submitted as a comment on PR

성공적으로 리소스에 반영되면 PR을 완료하고 병합(Merge)합니다.
작업을 완료하였으므로 feature branch는 삭제해줍니다.



Merge PR



Confirm merge

행복하십쇼