Terraform Secret Hands-On

tfvars, TFC Variable, SOPS+ephemeral+value_wo 비교 실습

BDACS

About Me

BDACS

김병훈

BDACS · DevOps Engineer

AWS Community Builder — Security

관심 분야

Terraform · **AWS** · K8S · DevOps · Security Infrastructure

LinkedIn

linkedin.com/in/byeonghunkim

발표를 준비하게 된 배경

런타임 권한은 열심히 분리했는데...

- IAM User - Policy
- EC2 Instance Profile
- ECS Execution Role / Task Role
- EKS IRSA / Pod Identity

```
resource "aws_ssm_parameter" "api_key" {
     name= "/demo/api-key"
     type= "SecureString"
     value = ???          # <-- ?
}
```

만약

1. 저장소(State)에 secret 평문으로 존재함

2. state가 공격자의 손에 들어가게 된다면?

-> 위의 권한에 대한 노력들에 대한 의미가 사라진다

그래서 **value**를 어떻게 안전하게 주입하지? 에서 궁금증 시작

목차

Terraform이란?

Infrastructure as Code (IaC)

인프라를 코드로 선언하고, 명령 한 줄로 생성/변경/삭제하는 도구

예시 : Terraform, CloudFormation, Pulumi

```
# 예: AWS에 S3 버킷 생성
resource "aws_s3_bucket" "my_bucket" {
    bucket = "my-app-bucket"
}

# 명령어 하나로 실제 AWS에 작성
# terraform apply
```

왜 Terraform?

- 수동 콘솔 작업 → 코드로 구현하여 자동으로 생성
- 변경 이력을 Git으로 추적 가능
- AWS, GCP, Azure 등 멀티 클라우드 지원

Terraform 동작 흐름

.tf 코드 작성
인프라를 HCL 언어로 선언

terraform login
TFC 인증 토큰 입력
(Terraform Cloud 를
backend로 쓸 때만 필요)

terraform init
Provider 플러그인 설치
+ Backend 연결
+ Module 설치

terraform plan
변경 사항 미리보기

terraform apply
실제 AWS에 반영
(Terrform Cloud를 쓰면
workspace 자격증명 등록)

Terraform Cloud

- Terraform의 State를 팀원과 공유하고, 원격에서 실행할 수 있게 해주는 SaaS 플랫폼
- HashCorp가 운영

State (terraform.tfstate)

- apply 결과를 기반으로 인프라 상태를 JSON으로 기록
- plan 실행 시 State와 코드를 비교하여 변경 사항 파악
- 협업 시 Backend (S3, TFC 등)에 안전하게 원격 저장

State는 어디에 저장되고, 누가 볼 수 있는가?

Local

terraform.tfstate
로컬 디스크에 파일로 저장

- ⓘ 파일 시스템 접근 = 열람
- ⓘ Git에 실수로 커밋 위험 - gitignore 설정
- ⓘ 혼자 파일을 가지고 있어 팀 협업 불가

S3 Backend

S3 버킷에 원격 저장
+ DynamoDB / S3 Native Lock

- ⓘ s3:GetObject 권한 = 열람
- ⓘ public access block 등
  버킷 정책 미설정 시, 외부 노출 가능
- ⓘ S3 SSE 설정에 따라 암호화 수준이 달라짐

Terraform Cloud

TFC 서버에 저장
암호화 at rest

- ⓘ S3 bucket 없이 간단한 poc 용이
- ⓘ 하지만 TFC API 토큰 관리 잘 해야함
- ⓘ 26년 4월 user 기반
  > resource 기반 가격정책 변경

State 파일은 실제로 이렇게 생겼습니다

```
// terraform.tfstate (JSON)
{
     "resources": [{
          "type": "aws_ssm_parameter",
          "name": "api_key",
          "instances": [{
               "attributes": {
                    "name": "/demo/api-key",
                    "type": "SecureString",
                    "value": "test1234" // <-- 평문!
               }
          }]
     }]
}
```

앞에서 말씀드린대로

apply 결과가

그대로 JSON에 기록

주의사항

State에 SecureString type이라도
시크릿이 **평문으로 기록될 수 있음**을 유의해야 함

오늘의 구성 정리

Provider (AWS)

```
terraform {
    required_providers {
        aws = { ... }
    }
}
provider "aws" {
    region = "ap-northeast-2"
}
```

어떤 클라우드의 어떤 리전에 API를 호출할지 설정
자격증명은 TFC - workspace에 진행
HCL 코드 → Terraform
→ AWS Provider (Go SDK) → AWS API

Backend (TFC)

```
terraform {
    cloud {
        organization = "<YOUR ORG>"
        workspaces {
            name = "<YOUR WKSP>"
        }
    }
}
```

State를 Terraform Cloud에 저장
팀 협업 + state 원격 저장

Resource SSM + KMS (case 3 only)

```
resource "aws_ssm_parameter" {
    name = "/demo/api-key"
    type = "SecureString"
    value = ???
}
```

실제 생성할 AWS 리소스
시크릿을 어떻게 넣을 것인가?

사전 준비 (1/3) — AWS IAM + CLI 설정

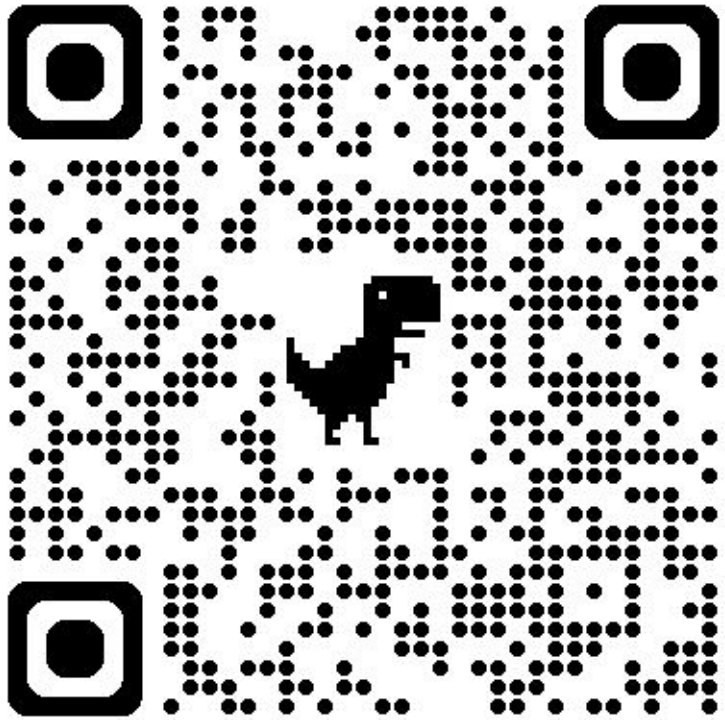
```
# 1. 레포 clone
git clone https://github.com/BASTION-Community/
      terraform-secret-workshop.git

# 2. 만약 Window시면, REAEME.md에 aws,terraform,sops
설치 가이드 링크 참조
```

3, IAM User 생성 (Administrator 또는 PowerUserAccess)
→ Access Key 발급 → AWS CLI 설정

```
# 4. AWS CLI 설치 + 프로필 등록
brew install awscli
aws --version                # 2.24.4
aws configure --profile secret-hands-on
export AWS_PROFILE=secret-hands-on
aws sts get-caller-identity
```

⚠️ 워크숍용 임시 키. 실습 후 반드시 삭제하세요.

또는 QR 코드로 repo 접근



사전 준비 (2/3) — Terraform 설치

```
# 1. Terraform 설치 (macOS)
brew tap hashicorp/tap
brew install hashicorp/tap/terraform
terraform -v        # v1.14.7
```

사전 준비 (3/3) — Terraform Cloud 설정

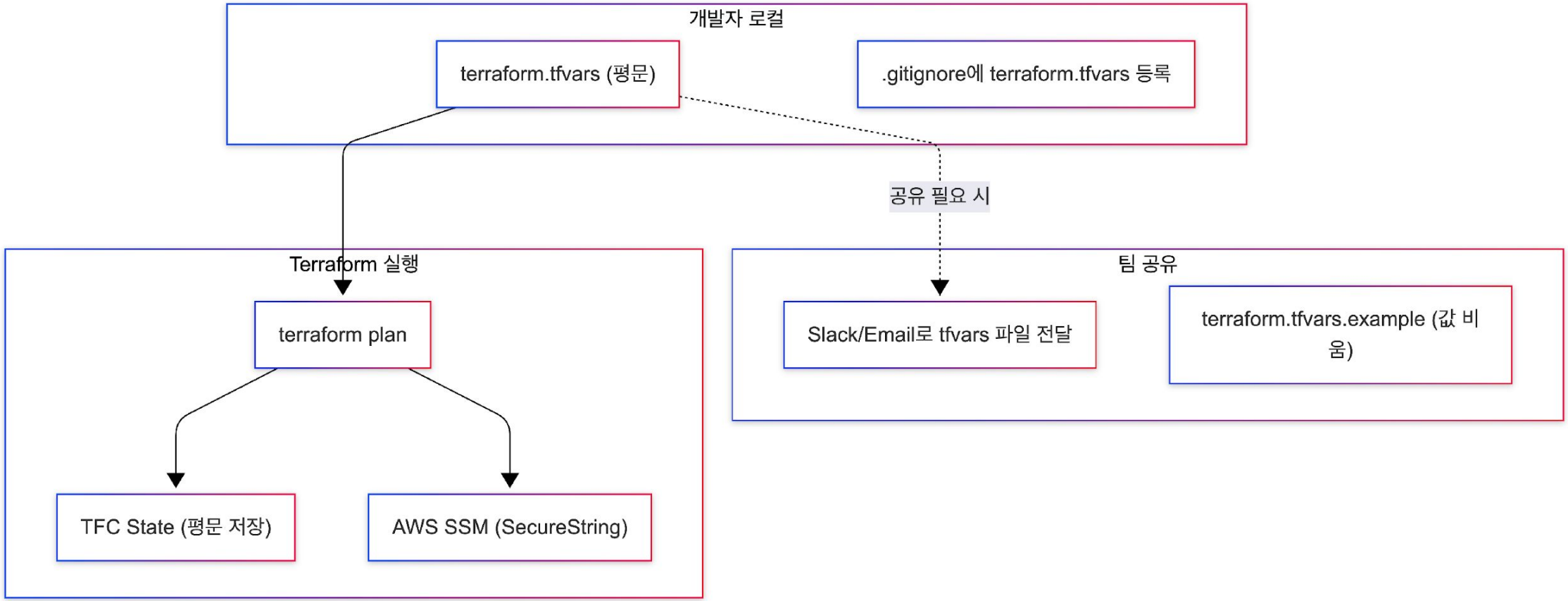
TFC 웹 설정 (app.terraform.io)

1. Organization 생성 (예: my-org)
2. Workspace 1개 생성 (예: secret-workshop)
3. Variables → Environment Variables에 AWS 키 등록:
   AWS_ACCESS_KEY_ID / AWS_SECRET_ACCESS_KEY

```
# 4. TFC 로그인
terraform login

# 5. 각 Case의 main.tf에서 YOUR_ORG / YOUR_WORKSPACE 수정
#    case1-tfvars/main.tf
#    case2-tfc-variable/main.tf
#    case3-sops/main.tf
```

⚠️ Access Key는 워크숍용 임시 키. 실습 후 반드시 삭제하세요.

Case 1: terraform.tfvars



Case 1: 구조

```
# variables.tf
variable "api_key" {}

# ssm.tf
resource "aws_ssm_parameter" "api_key" {
    name= "/demo/api-key"
    type= "SecureString"
    value = var.api_key
}
```

```
# terraform.tfvars
api_key = "test1234"

# .gitignore
*.tfvars
```

- tfvars에 시크릿을 평문으로 작성
- .gitignore로 Git에는 안 올라감
- **그런데... State는?**

Case 1: 핸즈온

HANDS-ON

실습 준비

case1-tfvars/ 디렉토리에서 README.md 기준으로 함께 실습하기

실습 실행

tfvars에 시크릿을 넣고 apply 수행

Case 1: 돌아보기

OK Git에는 안 올라간다 (.gitignore 설정)

OK SSM에는 암호화 저장 (SecureString 타입 사용)

BUT 로컬에 평문 파일이 남음 (terraform.tfvars)

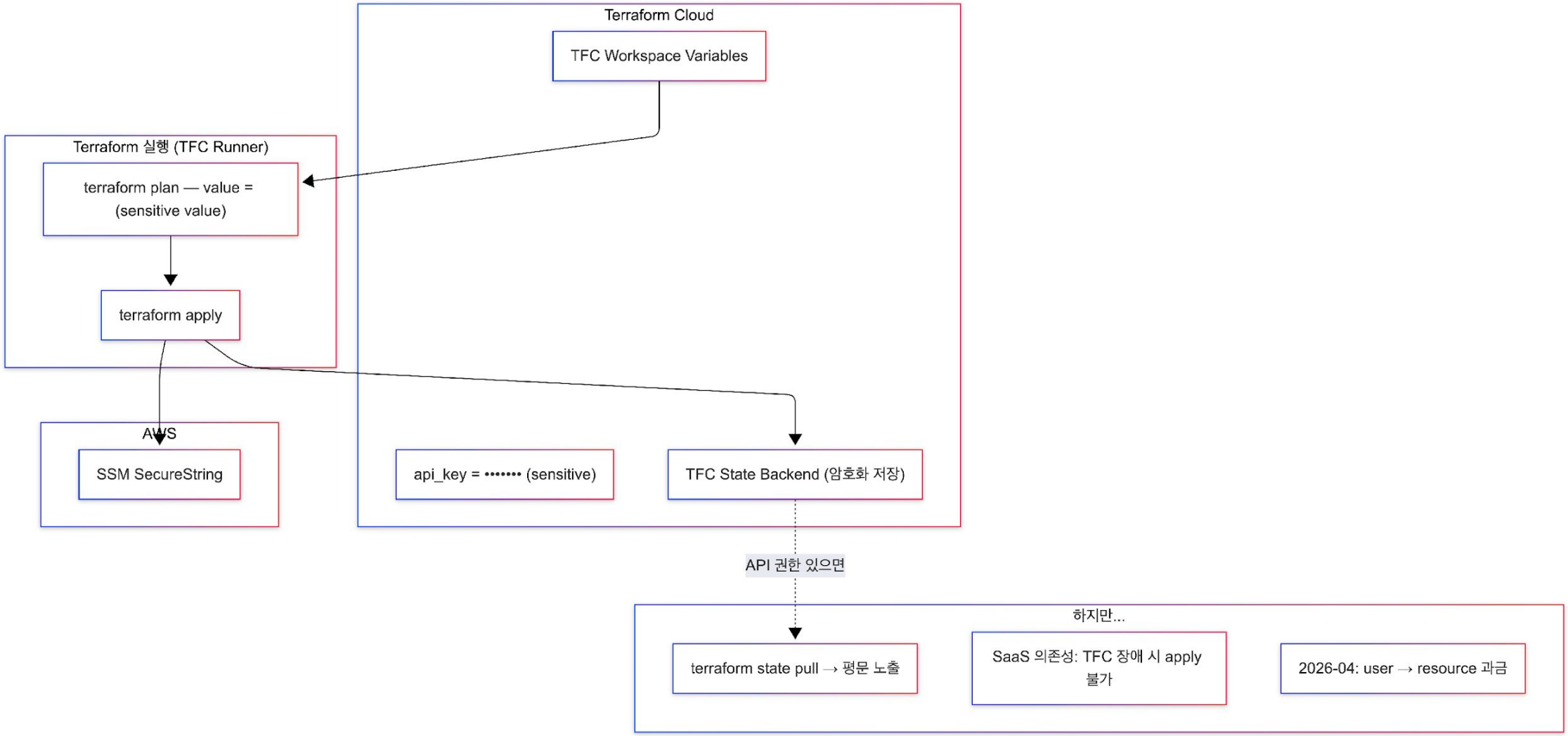
BUT **State에 시크릿이 평문으로 기록된다**

Case 1: 핸즈온 마무리

HANDS-ON

```
# 1. case1 디렉토리 에서 terraform destory
terraform destroy -auto-approve
```

Case 2: TFC UI에서 넣자



sensitive = true는 무엇을 하는가?

```
variable "api_key" {
    sensitive = true
}
```

✅ 하는 것

- plan/apply CLI 출력에서 값 마스킹
- terraform output에서 값 숨김

❌ 안 하는 것

- State 파일 암호화
- State에서 평문 제거
- API 로그에서 값 제거

```
# plan 출력 – 마스킹됨
+ value = (sensitive value)

# state show – 마스킹됨
value = (sensitive value)

# state pull – 평문!
"value": "test1234"
```

가려주는 것 ≠ 보호하는 것

Case 2: 무엇이 달라지나

```hcl
# variables.tf
variable "api_key" {
    sensitive = true # CLI 출력 마스킹
}

# ssm.tf — Case 1과 동일
resource "aws_ssm_parameter" "api_key" {
    name = "/demo/api-key"
    type = "SecureString"
    value = var.api_key
}
```

1. terraform.tfvars 없음

· TFC UI에서 Sensitive Variable로 설정

2. sensitive = true

· plan/apply 출력에서 값이 마스킹됨

그런데 State는?

Case 2: 핸즈온

HANDS-ON

- case2-tfc-variable/ 디렉토리에서 README.md 기준으로 함께 실습하기
- TFC UI에서 sensitive variable 설정 후 apply

주목할 포인트

Q. sensitive = true인데 state pull에서 여전히 평문?

Case 1과 결과가 같다 — 전달 방식을 바꿔도 State는 그대로

Case 2: 핸즈온 마무리

HANDS-ON

```
# case2 디렉토리에서 terraform destroy
terraform destroy -auto-approve
```

Case 1 vs Case 2

	Case 1 (tfvars)	Case 2 (TFC Variable)
시크릿 전달	로컬 파일	TFC UI
로컬 디스크	BUT 평문 파일	OK 파일 없음
State	BUT 평문	BUT 여전히 평문

전달 방식을 바꿔도, value = var.api_key 인 한 State 문제는 해결되지 않는다

쉬어가기 : Vercel NHI 보안 침해 (2026.04)

NHI (Non-Human Identity): OAuth 토큰, API 키 등 사람이 아닌 시스템의 인증 정보

1. 3rd-party 침해

Google Workspace
OAuth 앱 침해

→

2. OAuth 권한 타고 측면 이동

Vercel 내부
GitHub 저장소 접근

→

3. 데이터 탈취

접근 키 유출
DB 데이터 · API 키

→

4. 판매 시도

해킹 포럼에
판매 게시

Vercel 자체가 뚫린 게 아니라, Vercel이 쓰던 3rd-party OAuth 앱이 먼저 침해됨

- 조직의 공격면 = **승인한 OAuth 앱들의 합집합**. OAuth 앱 하나가 전사 SPOF
- 환경변수는 유출 즉시 **로테이션 (Rotation)** 권고

만약 case2 를 Vercel 사건에 대입해본다면 ?

Vercel 사건 (실제)

- 3rd-party AI 도구 OAuth 침해
- OAuth 권한으로 GitHub 접근
- non-sensitive 환경변수 유출
- 데이터 탈취

TFC에 대입하면

- 3rd-party 침해 → TFC API 토큰 탈취
- TFC 변수 + State 접근
- **sensitive/write-only여도**
  내부 저장은 복호화 가능
- **State에 평문 시크릿이 있으면 전부 노출**

UI에서 안 보이는 것(sensitive) ≠ 안전한 것

잠시 휴식

쉬었다가 Case 3로 이어가겠습니다

Case 3: Zero-Secret 패턴

SOPS + Ephemeral + value_wo

SOPS란?

Secrets OPerationS (SOPS)

시크릿 파일을 암호화해서 Git에 안전하게 커밋할 수 있게 해주는 도구

GitHub 21.5k+ stars

Community Trusted

CNCF Sandbox Project

(2023.05 Adopted)

Multi-KMS Support

KMS, GCP KMS, Azure Key Vault, age, PGP

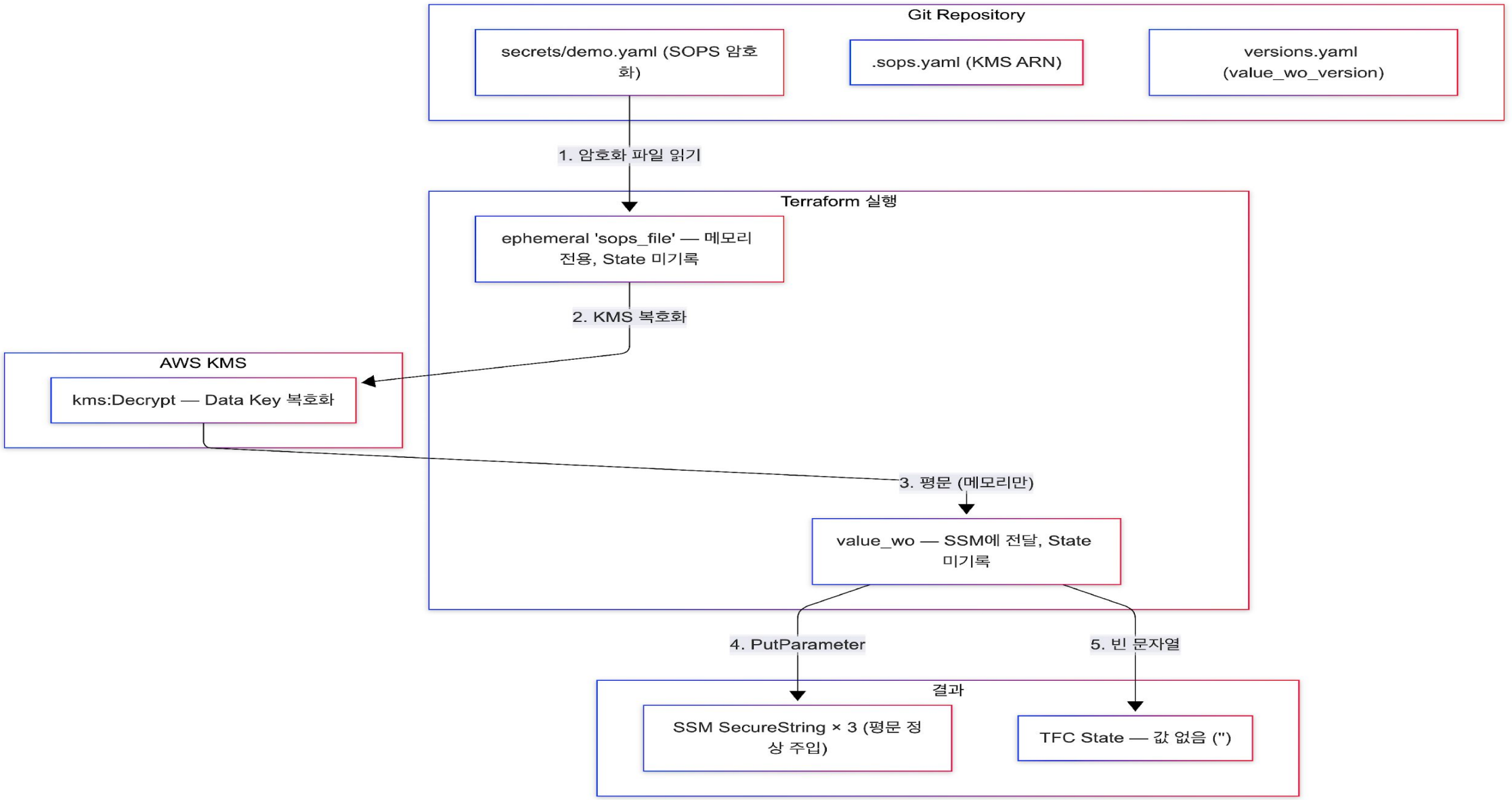
GitHub

github.com/getsops/sops

CNCF

cncf.io/projects/sops

Case 3: SOPS + Ephemeral + value_wo



Case 3의 핵심 아이디어: 3가지 조합

SOPS

시크릿을 암호화해서

Git에 안전하게 저장

ephemeral

복호화된 값을

메모리에만 보관

State에 기록 안 함

value_wo

AWS API에 전달만 하고

State에는 빈 문자열 저장

Case 3: 코드 구조

```
# kms.tf
resource "aws_kms_key" "sops" {
  description         = "SOPS encryption"
  enable_key_rotation = true
}
```

KMS 키 — SOPS 암호화/복호화에 사용

```
# ssm.tf — 핵심 코드
ephemeral "sops_file" "demo" {
  source_file = "${path.module}/secrets/demo.yaml"
}

resource "aws_ssm_parameter" "demo" {
  for_each         = local.versions
  name             = each.key
  type             = "SecureString"
  value_wo         = ephemeral.sops_file.demo.data[each.key]
  value_wo_version = each.value
}
```

Case 3 핸즈온

HANDS-ON

01. 환경 준비

SOPS 설치:
`brew install sops`

// sops 3.12.1

02. 가이드 확인

`case3-sops/` 디렉토리의 **README.md** 기준으로 함께 실습하기

03. 핵심 워크플로우

KMS 키 생성
↓
SOPS 암호화
↓
apply → State 확인

주목할 포인트

- apply 로그: `Opening ephemeral... Closing ephemeral...`
- state pull → **value, value_wo 모두 빈 문자열!**
- SSM에는 실제 값이 들어가 있는가?
- **State에는 빈 문자열, AWS에는 실제 값 = Zero-Secret**

value_wo의 트레이드오프 — 값 변경 감지

STEP 01. SOPS 파일에서 값 수정

```
# SOPS 파일에서 값을 변경한 뒤
$ sops secrets/demo.yaml # 에디터에서 값 수정 후 저장

$ terraform plan
# -> No changes. Your infrastructure matches the configuration.
```

STEP 02. versions.yaml 버전 업데이트

```
# versions.yaml에서 해당 키의 버전을 bump
# "/demo/api-key-1": 1 -> "/demo/api-key-1": 2

$ terraform plan
# ~ aws_ssm_parameter.demo["/demo/api-key-1"]
# ~ value_wo_version = 1 -> 2 <- 이제 변경 감지!
```

- **value_wo는 State에 존재하지 않아 Terraform이 이전 값을 알 수 없습니다.**
- versions.yaml의 버전 수정을 통해 명시적으로 변경을 알려야 합니다. (Zero-Secret의 트레이드오프)

Case 3 핸즈온 마무리

HANDS-ON

```
# case3 디렉토리에서 terraform destroy
terraform destroy -auto-approve
```

3 Case 비교

	Case 1 (tfvars)	Case 2 (TFC)	Case 3 (SOPS)
시크릿 전달	로컬 파일	TFC UI	암호화 파일
로컬 디스크	BUT 평문	OK 없음	OK 암호문
Git 저장	BUT 불가	해당없음	OK 커밋 가능
State	BUT 평문	BUT 평문	OK 빈 문자열
변경 감지	자동	자동	수동 (version)

Deep Dive: SOPS는 어떻게 암호화하나?

Envelope Encryption

SOPS 암호화 파일의 내부

```
# secrets/demo.yaml (암호화된 상태)
api_key: ENC[AES256_GCM,data:abc...,
         iv:xxxx,tag:yyyy,type:str]

db_pass: ENC[AES256_GCM,data:xyz...,
         iv:zzzz,tag:wwww,type:str]

sops:
    kms:
        - arn: arn:aws:kms:...:key/xxx
          enc: AQICAHh... # Wrapped Data Key
```

값 (value)

AES-256-GCM으로 암호화됨

키 이름 (key)

평문 — 주의 필요 (노출됨)

enc 필드

KMS로 암호화된 Data Key

Envelope Encryption

Lock 1: AES-256-GCM

각 값을 Data Key로 암호화

Data Key = 32바이트 랜덤 키 (로컬 생성)

Lock 2: KMS CMK

Data Key 자체를 KMS CMK로 암호화

Managed Key (aws/ssm)는 Key Policy Custom 불가, SOPS 사용 불가

Data Key 생성 방식

- crypto/rand로 32바이트 로컬 생성
- kms:GenerateDataKey API 호출 안 함
- kms:Encrypt로 래핑만 요청

왜 봉투 암호화인가?

- KMS API 1번만 호출 (값이 100개여도)
- KMS 4KB 제한 우회
- 다른 키 서비스로 교체 가능
  (GCP KMS, Azure, etc)

sops -d 복호화 흐름

1. 파일에서 래핑된 Data Key 추출 (enc: AQICAHh...)

2. AWS KMS API 호출: kms:Decrypt

3. KMS가 HSM 내부에서 복호화 → 평문 Data Key 반환

4. Data Key로 각 값을 AES-256-GCM 복호화

5. 평문 출력 + 메모리에서 Data Key 삭제

KMS 키 정책 = 시크릿 접근 정책

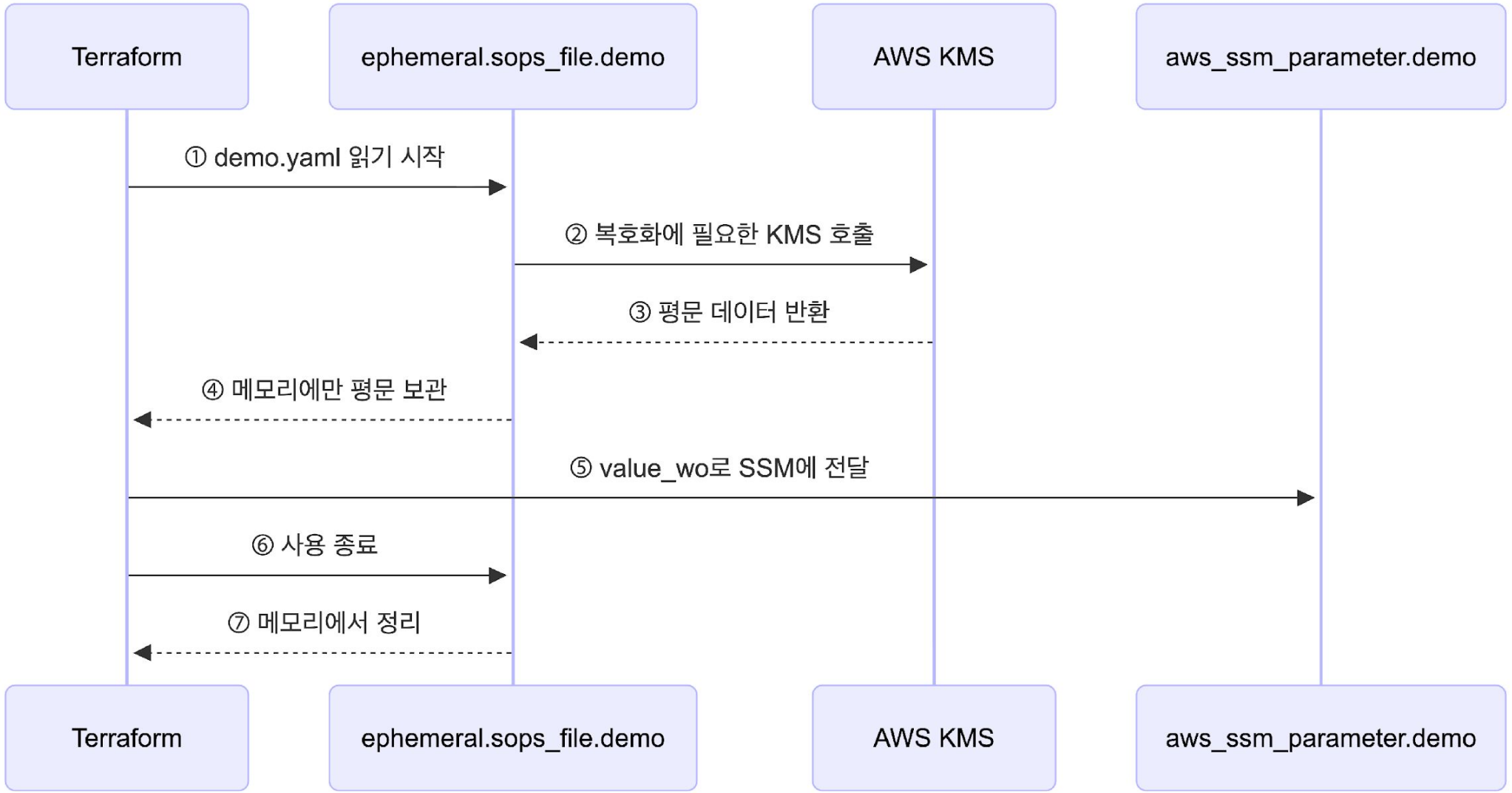
AES-256-GCM

256비트 키로 암호화 + 변조 여부 자동 탐지

구성 요소	역할
AES	대칭 암호화 표준
256	키 길이 256bit (brute force 불가능)
GCM	암호화 + 밀랍 씰 — 뜯으면 복호화 거부

Git에서 암호화 파일을 변조하면? → Tag 검증 실패 → 복호화 거부

ephemeral + value_wo 동작 흐름



정리: Zero-Secret 패턴

SOPS

시크릿을 KMS로 암호화하여
Git에 안전하게 저장

ephemeral

복호화 값은 메모리에만,
State에 기록하지 않음

value_wo

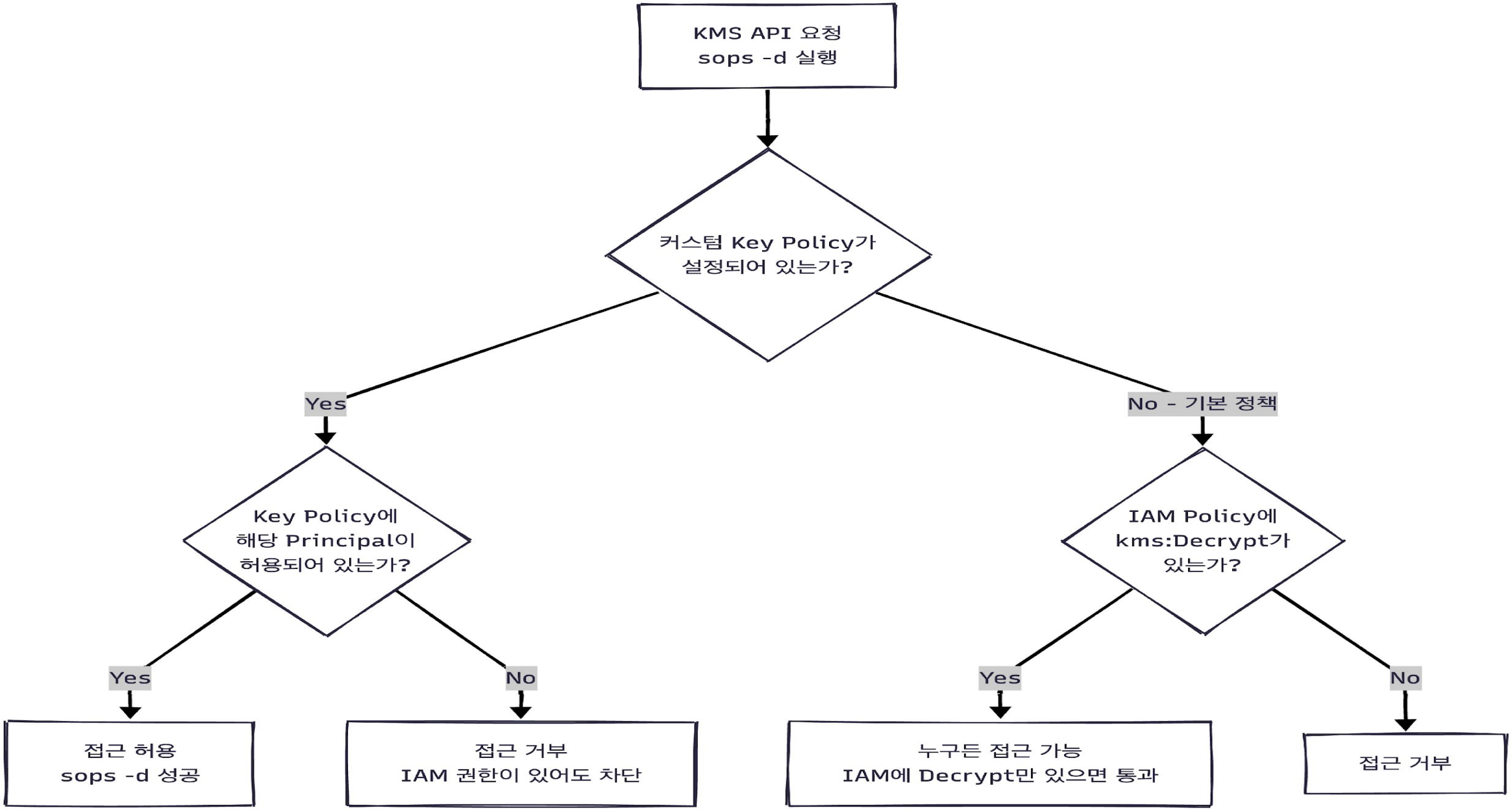
AWS API에 전달만,
State에는 빈 문자열

어디에도 평문이 디스크에 남지 않는다 = Zero-Secret

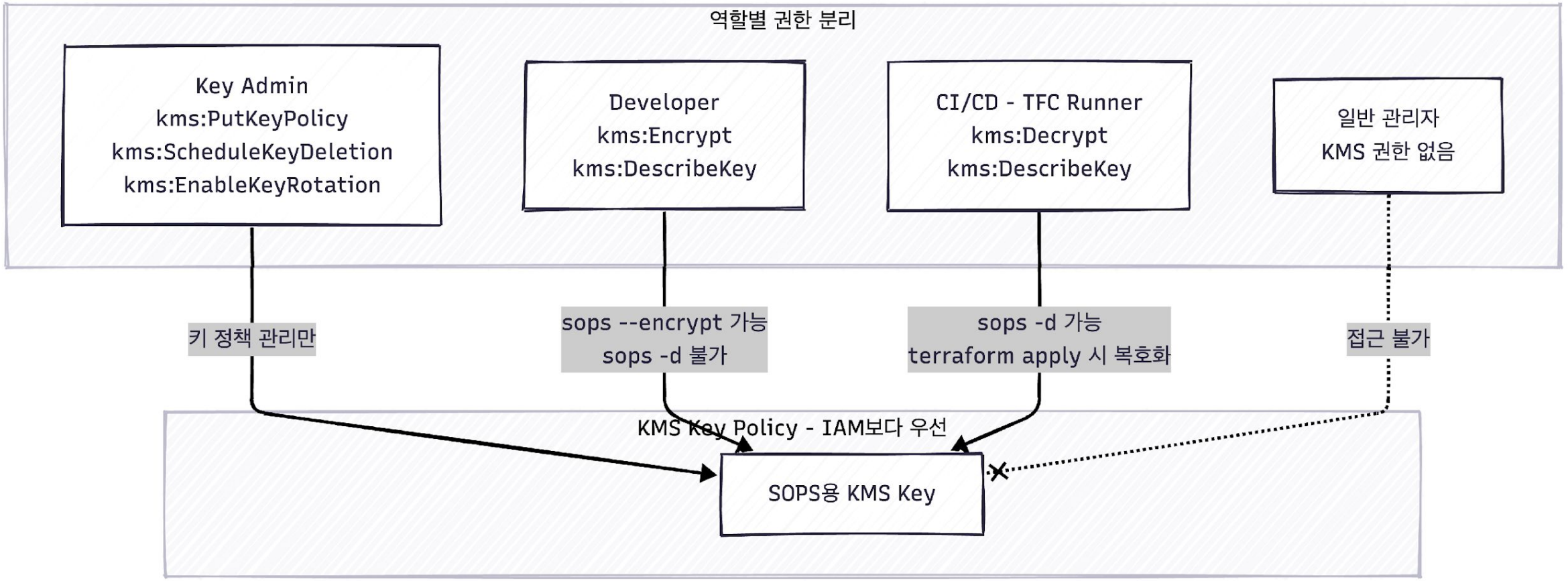
더 나아가기

좀 더 도전해볼 수 있는 내용들

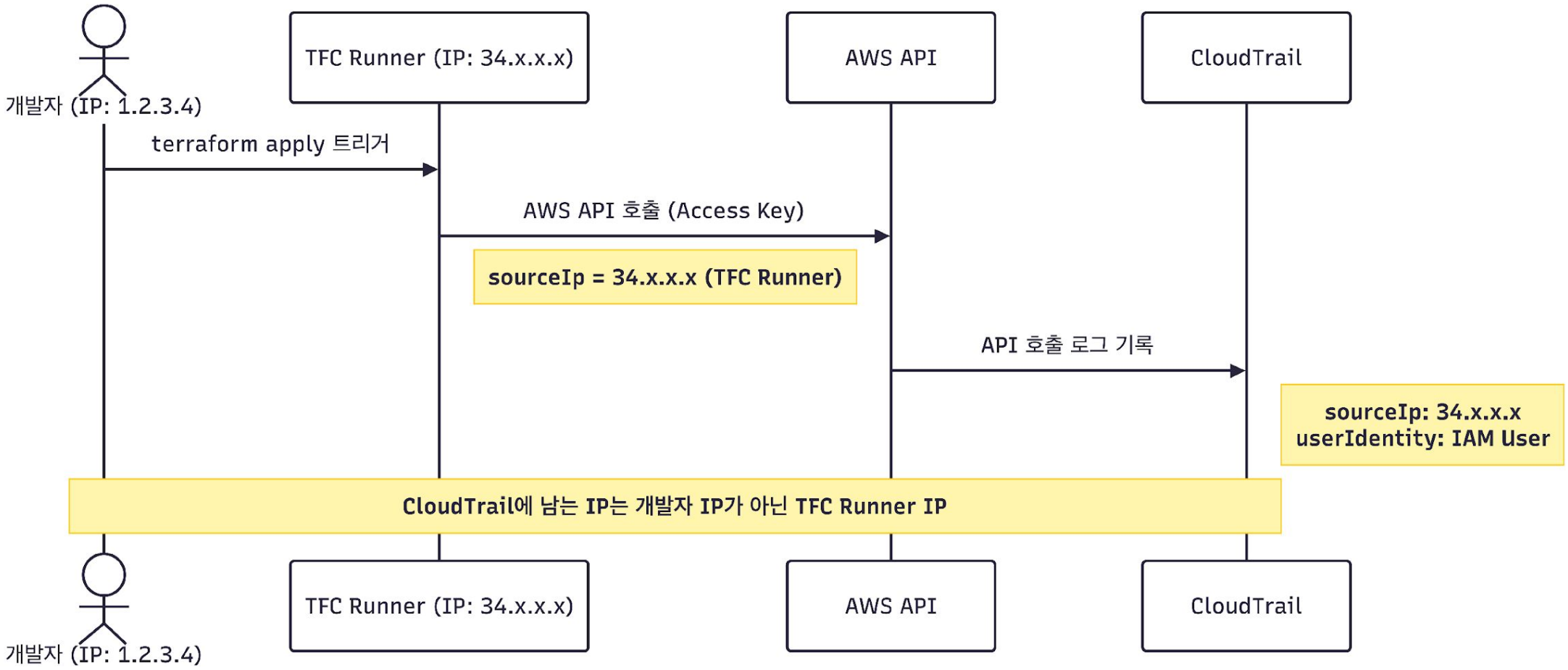
KMS Key Policy가 없으면?



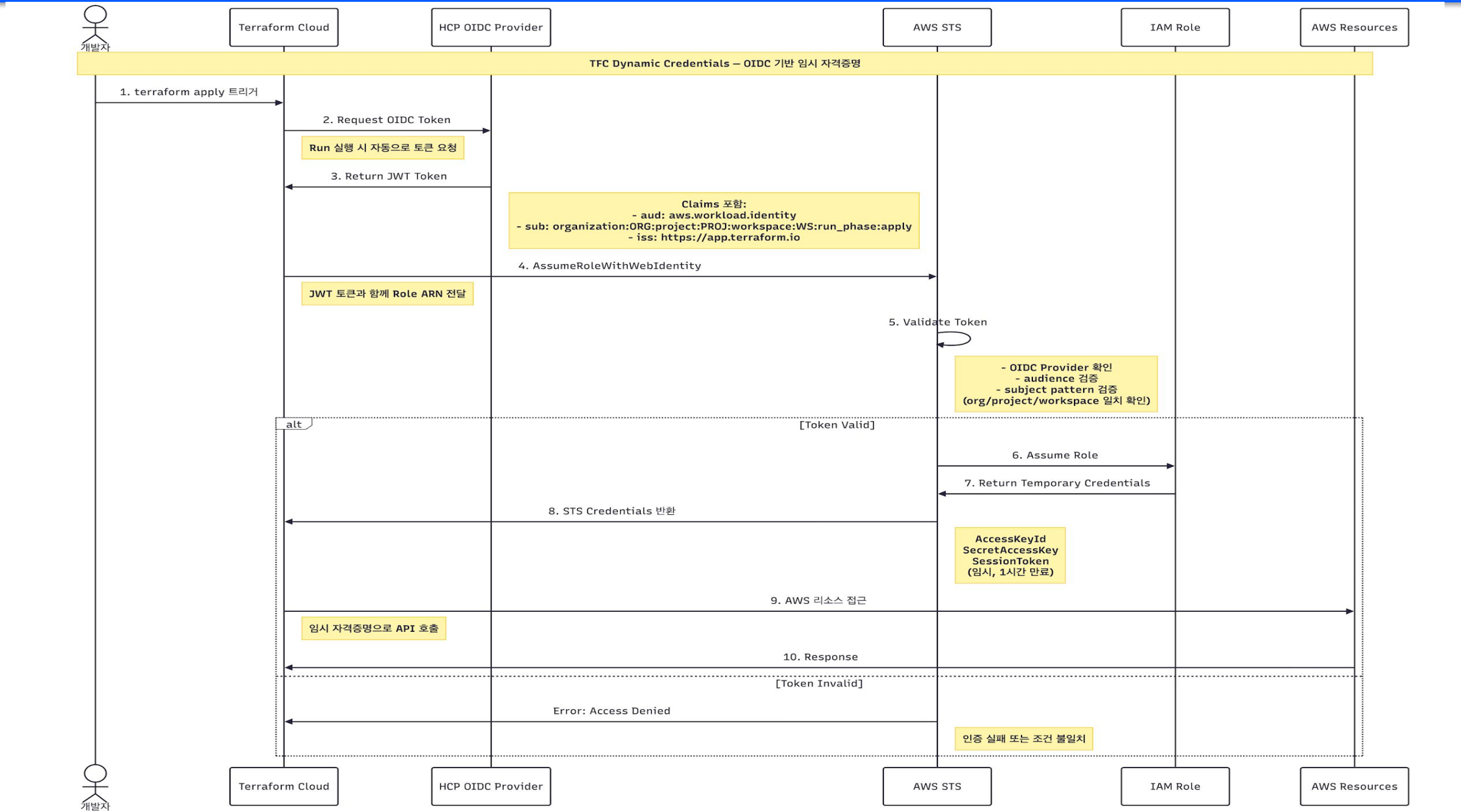
KMS Key Policy — 역할 분리



Access Key를 TFC에 넣으면 생기는 문제



해결 : TFC OIDC — Dynamic Credentials



Q & A

고생하셨습니다.

BDACS