DEVIEW 2021

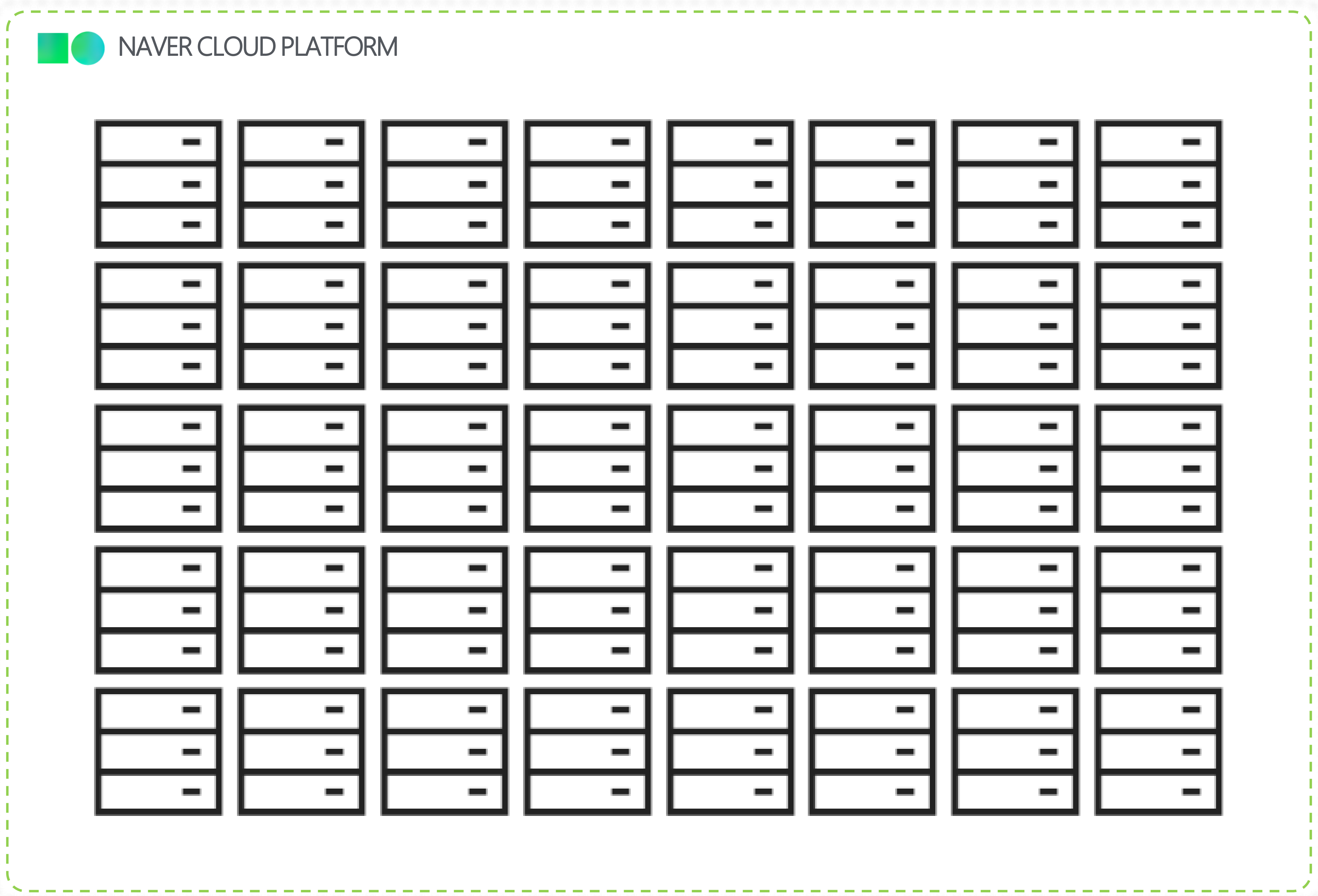
네이버 클라우드 플랫폼 Terraform Provider 개발기

김상규 NAVER Cloud

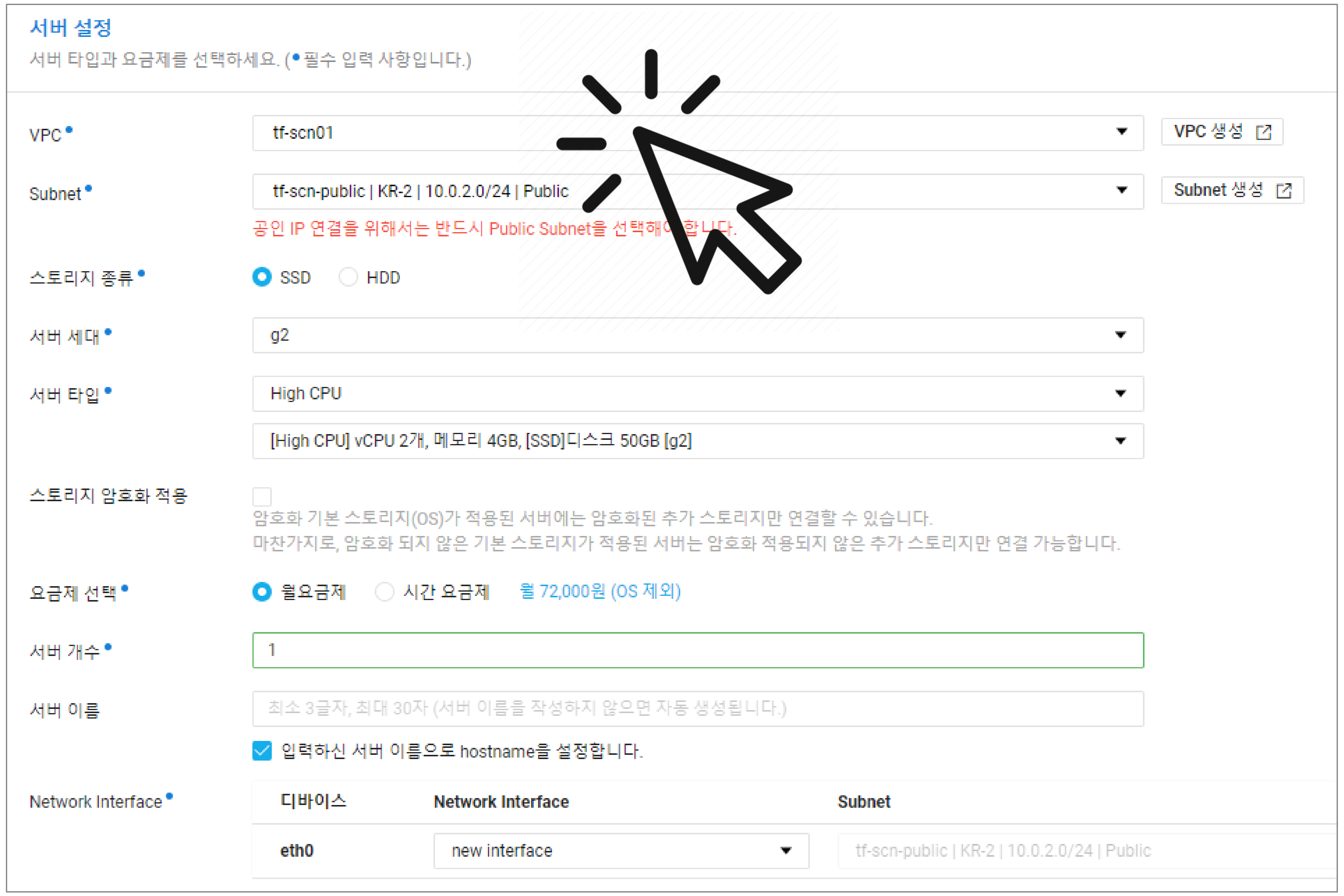
Prologue

Terraform 을 필요로 하는 고객들의 요청

코드 vs 마우스 클릭



NAVER CLOUD PLATFROM 콘솔



서버 설정

서버 타입과 요금제를 선택하세요. (• 필수 입력 사항입니다.)

VPC •: tf-scn01 ▾ VPC 생성

Subnet •: tf-scn-public | KR-2 | 10.0.2.0/24 | Public ▾ Subnet 생성

공인 IP 연결을 위해서는 반드시 Public Subnet을 선택해야 합니다.

스토리지 종류 •: ● SSD ○ HDD

서버 세대 •: g2 ▾

서버 타입 •: High CPU ▾

[High CPU] vCPU 2개, 메모리 4GB, [SSD]디스크 50GB [g2] ▾

스토리지 암호화 적용: □

암호화 기본 스토리지(OS)가 적용된 서버에는 암호화된 추가 스토리지만 연결할 수 있습니다.
마찬가지로, 암호화 되지 않은 기본 스토리지가 적용된 서버는 암호화 적용되지 않은 추가 스토리지만 연결 가능합니다.

요금제 선택 •: ● 월요금제 ○ 시간 요금제 월 72,000원 (OS 제외)

서버 개수 •: 1

서버 이름: 최소 3글자, 최대 30자 (서버 이름을 작성하지 않으면 자동 생성됩니다.)

☑ 입력하신 서버 이름으로 hostname을 설정합니다.

Network Interface •

디바이스	Network Interface	Subnet
eth0	new interface ▾	tf-scn-public | KR-2 | 10.0.2.0/24 | Public

Prologue

Terraform 을 필요로 하는 고객들의 요청

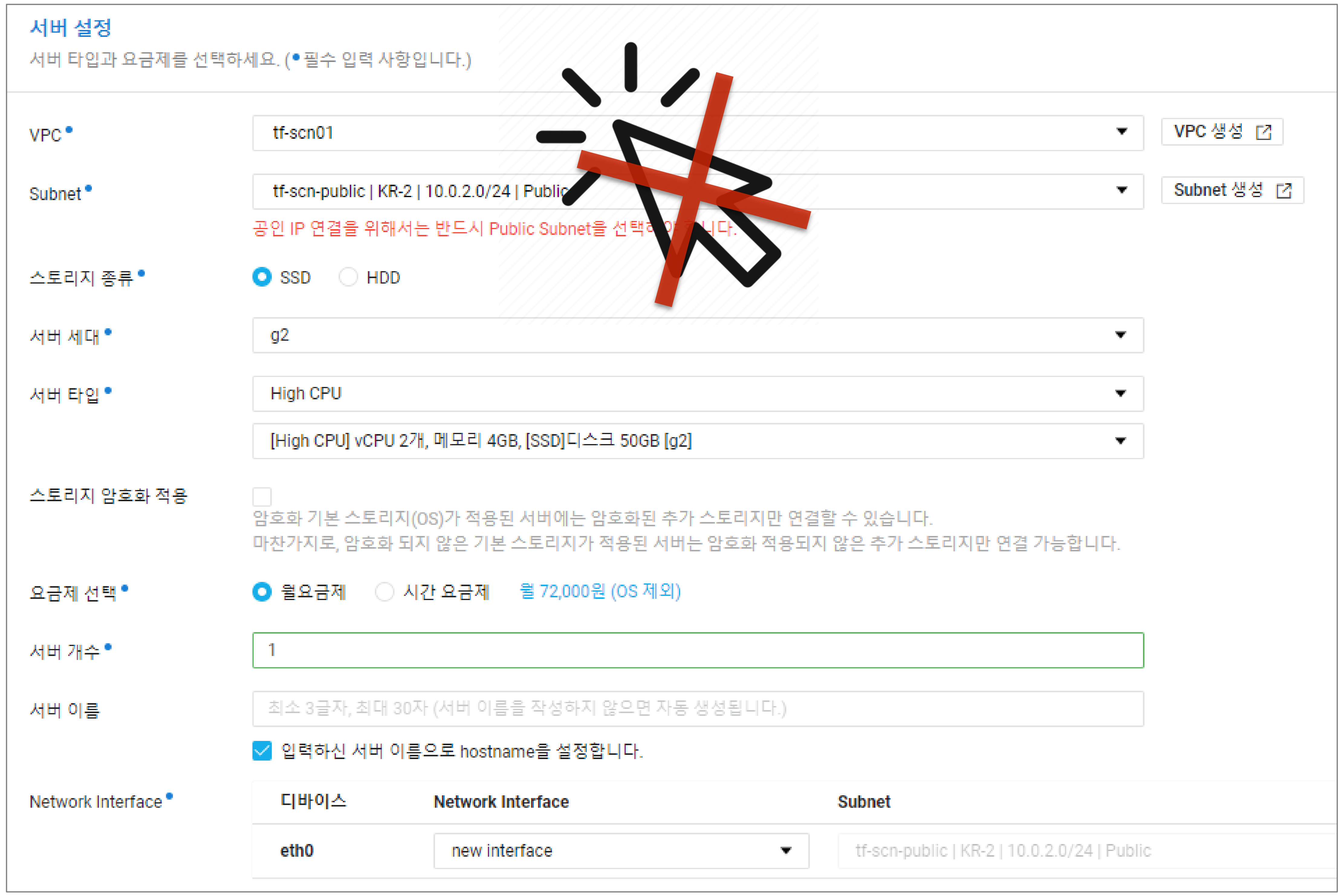
코드 vs 마우스 클릭

terraform code

```
resource "ncloud_server" "server" {
  subnet_no                  = ncloud_subnet.test.id
  name                       = "my-tf-server-${count.index}"
  server_image_product_code = "SW.VSVR.OS.LNX64.CNTOS.0703.B050"
  count                      = 100
}
```

VS

NAVER CLOUD PLATFROM 콘솔



서버 설정

서버 타입과 요금제를 선택하세요. (•필수 입력 사항입니다.)

VPC• tf-scn01 ▾ VPC 생성

Subnet• tf-scn-public | KR-2 | 10.0.2.0/24 | Public ▾ Subnet 생성

공인 IP 연결을 위해서는 반드시 Public Subnet을 선택해야 합니다.

스토리지 종류• ● SSD ○ HDD

서버 세대• g2 ▾

서버 타입• High CPU ▾

[High CPU] vCPU 2개, 메모리 4GB, [SSD]디스크 50GB [g2] ▾

스토리지 암호화 적용 □

암호화 기본 스토리지(OS)가 적용된 서버에는 암호화된 추가 스토리지만 연결할 수 있습니다.
마찬가지로, 암호화 되지 않은 기본 스토리지가 적용된 서버는 암호화 적용되지 않은 추가 스토리지만 연결 가능합니다.

요금제 선택• ● 월요금제 ○ 시간 요금제 월 72,000원 (OS 제외)

서버 개수• 1

서버 이름 최소 3글자, 최대 30자 (서버 이름을 작성하지 않으면 자동 생성됩니다.)

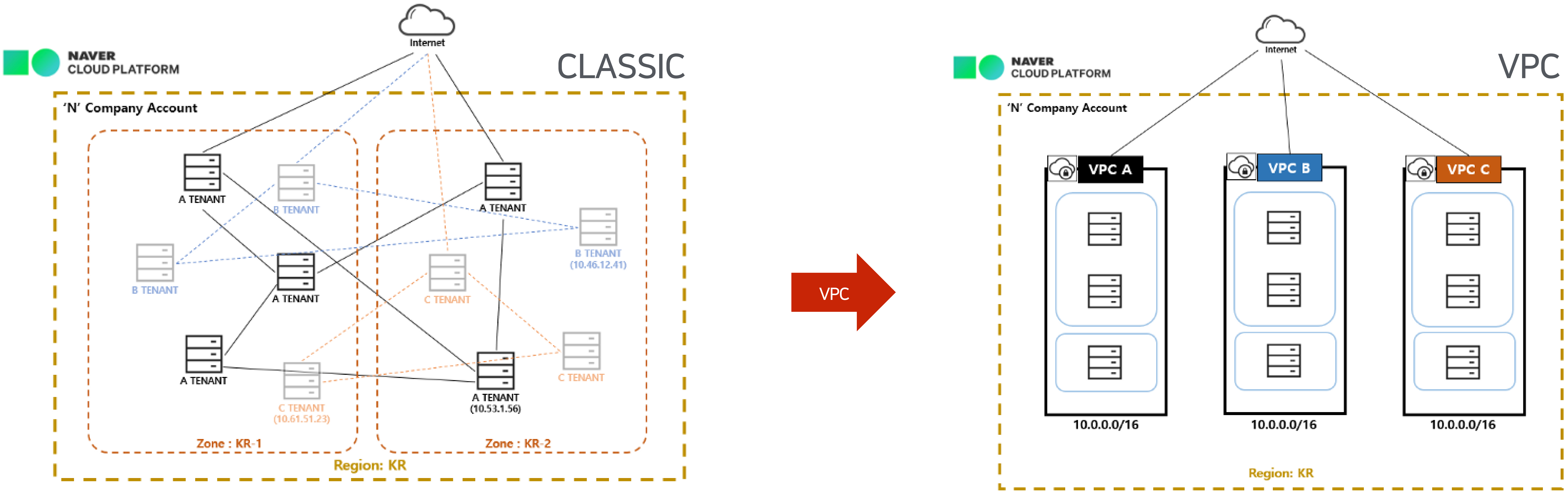
☑ 입력하신 서버 이름으로 hostname을 설정합니다.

Network Interface•

디바이스	Network Interface	Subnet
eth0	new interface ▾	tf-scn-public | KR-2 | 10.0.2.0/24 | Public

Prologue

2020년 9월 VPC 출시

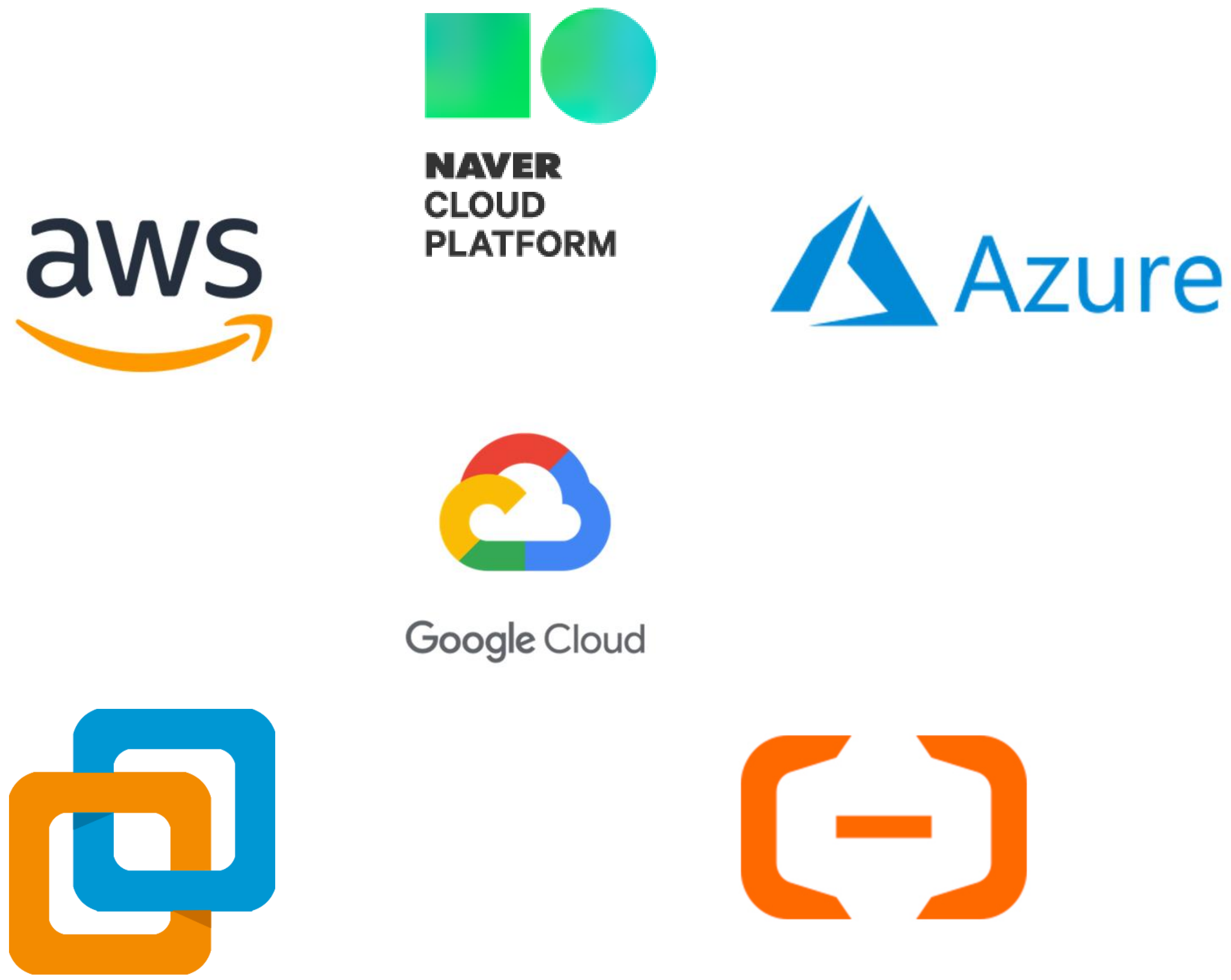


CONTENTS

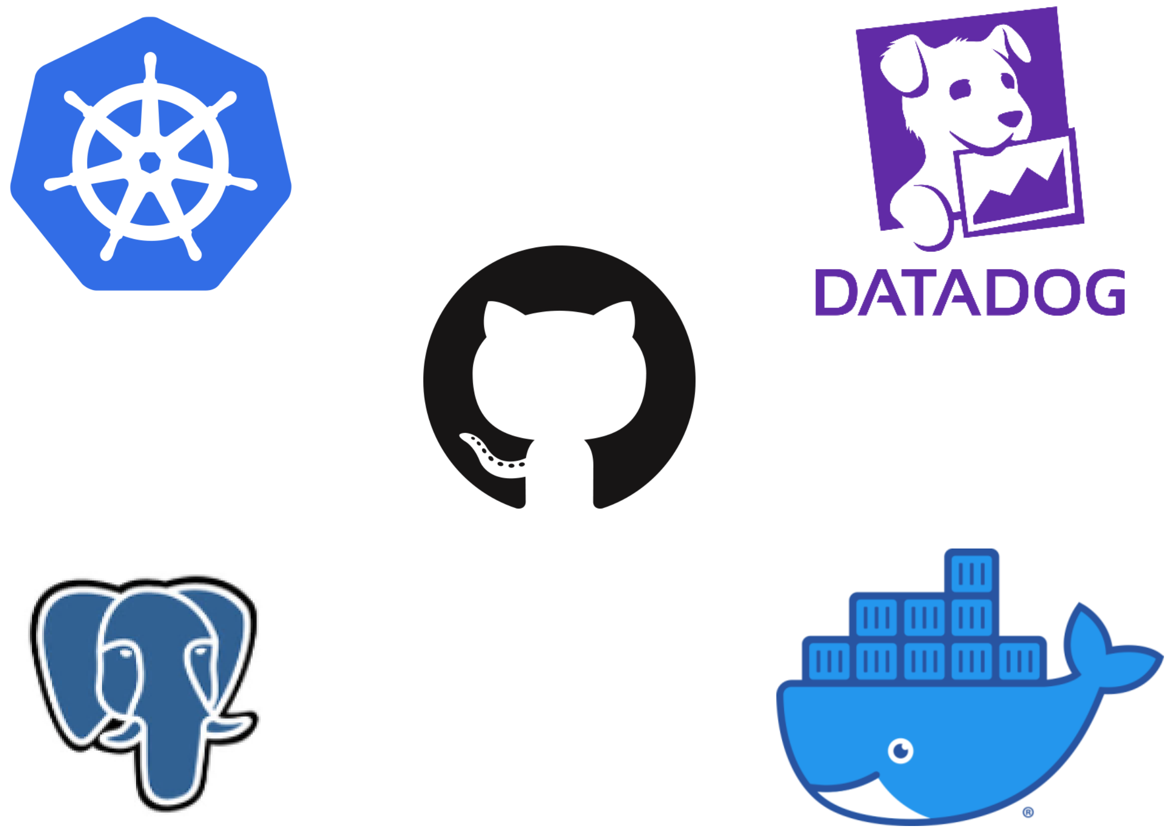
1. Terraform providers?
2. How terraform works
3. Terraform provider 만들기
4. Appendix & Tips
5. Summary

Terraform providers?

Cloud providers



PaaS, SaaS providers



other APIs

Terraform providers?

Cloud providers

NAVER
CLOUD
PLATFORM

```
resource "ncloud_vpc" "vpc" {
  ipv4_cidr_block = "10.0.0.0/16"
}

resource "ncloud_subnet" "pub-sub" {
  vpc_no         = ncloud_vpc.vpc.id
  subnet         = "10.0.1.0/24"
  zone           = "KR-2"
  network_acl_no = ncloud_vpc.vpc.default_network_acl_no
  subnet_type    = "PUBLIC"
}

resource "ncloud_server" "server" {
  subnet_no                 = ncloud_subnet.pub-sub.id
  name                      = "my-tf-server"
  server_image_product_code = "SW.VSVR.OS.LNX64.CNTOS.0703.B050"
}
```

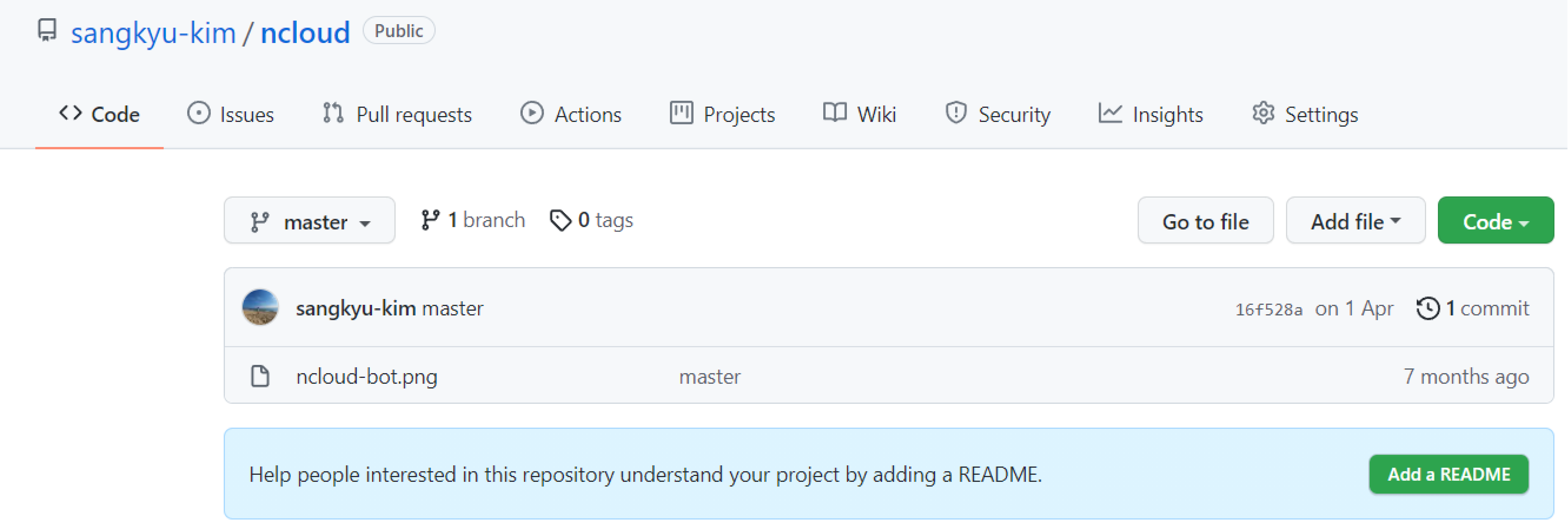
Terraform providers?

PaaS, SaaS providers

```
resource "kubernetes_service" "example" {
  metadata {
    name = "terraform-example"
  }
  spec {
    selector = {
      app = kubernetes_pod.example.metadata.0.labels.app
    }
    session_affinity = "ClientIP"
    port {
      port        = 8080
      target_port = 80
    }

    type = "LoadBalancer"
  }
}
```

Terraform providers?

PaaS, SaaS providers



sangkyu-kim master 16f528a on 1 Apr 1 commit

ncloud-bot.png master 7 months ago

Help people interested in this repository understand your project by adding a README.

```
resource "github_repository" "example" {
  name        = "ncloud"
  description = "My awesome codebase"

  visibility = "public"

  template {
    owner      = "github"
    repository = "terraform-module-template"
  }
}
```

Terraform providers?

Other APIs

```
resource "hue_light" "example" {
  unique_id = "00:17:88:01:03:97:02:b8-0b"

  state {
    hue = 24918
    on  = true
  }
}
```

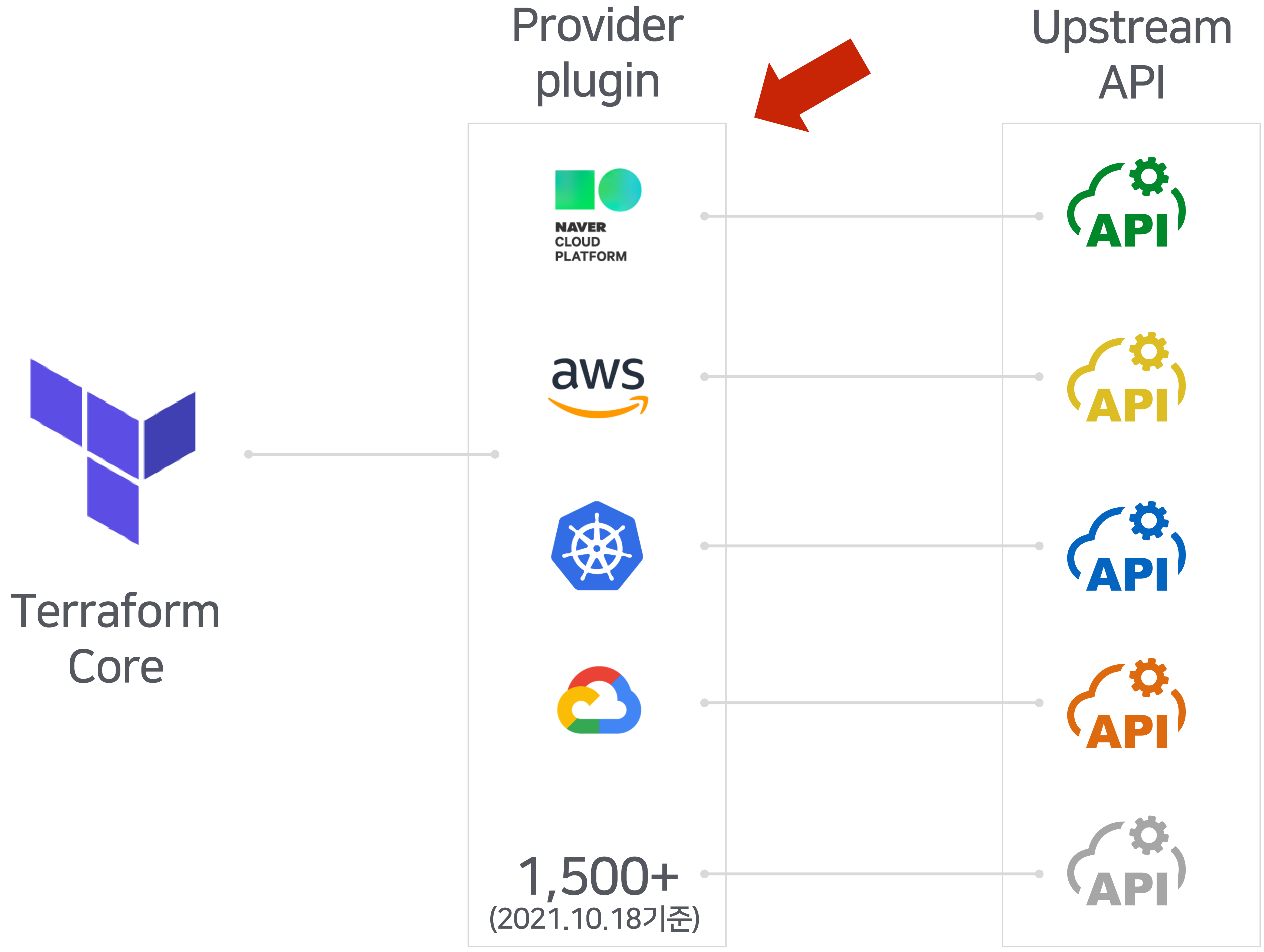
Terraform providers?

Other APIs

```
resource "hue_light" "example" {
  unique_id = "00:17:88:01:03:97:02:b8-0b"

  state {
    hue = 24918
    on  = false
  }
}
```

Terraform providers?

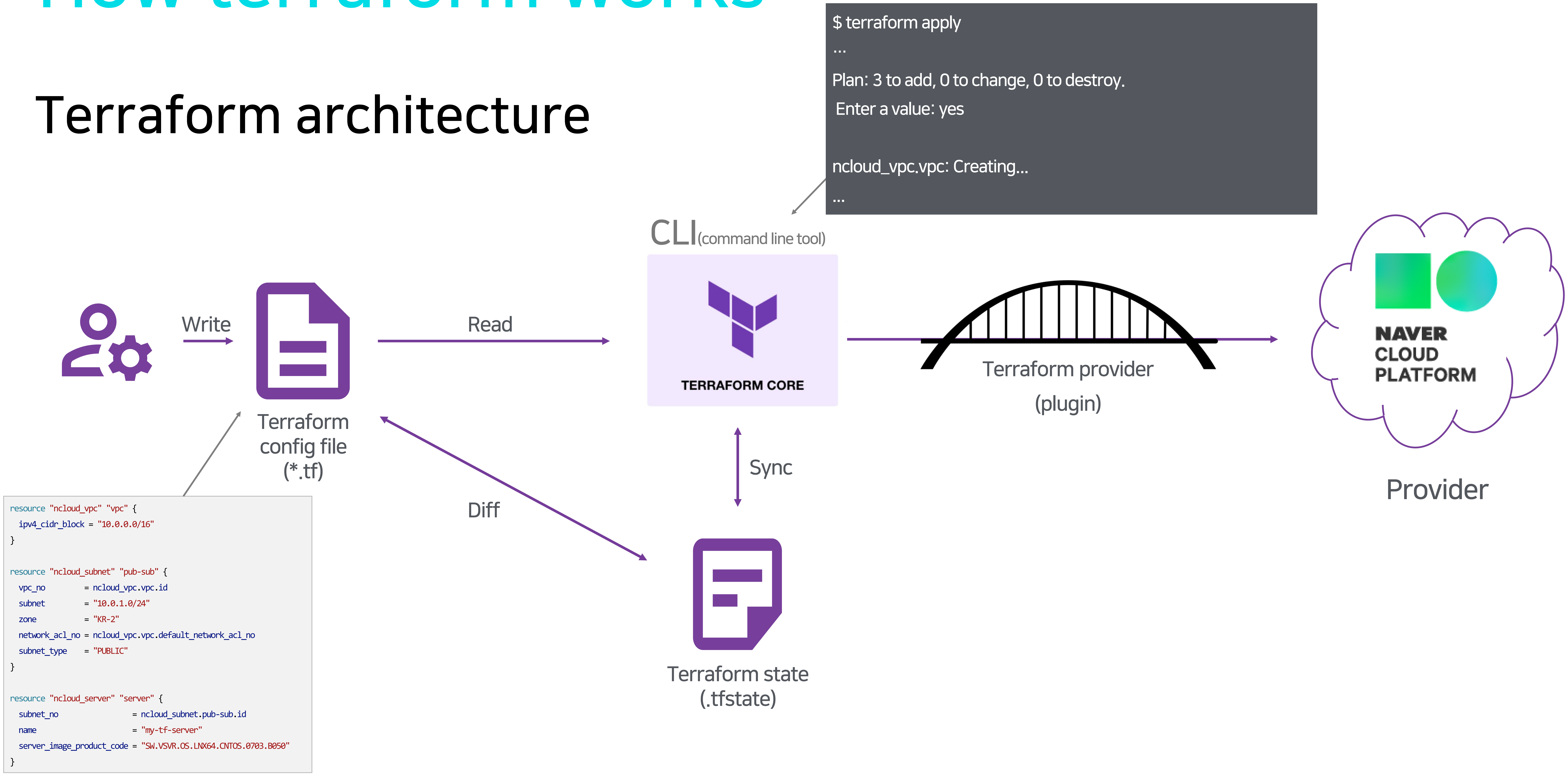
1,000개가 넘는 Terraform providers



2. How terraform works

How terraform works

Terraform architecture



```
$ terraform apply
...
Plan: 3 to add, 0 to change, 0 to destroy.
 Enter a value: yes

ncloud_vpc.vpc: Creating...
...
```

```
resource "ncloud_vpc" "vpc" {
  ipv4_cidr_block = "10.0.0.0/16"
}

resource "ncloud_subnet" "pub-sub" {
  vpc_no         = ncloud_vpc.vpc.id
  subnet         = "10.0.1.0/24"
  zone           = "KR-2"
  network_acl_no = ncloud_vpc.vpc.default_network_acl_no
  subnet_type    = "PUBLIC"
}

resource "ncloud_server" "server" {
  subnet_no                 = ncloud_subnet.pub-sub.id
  name                      = "my-tf-server"
  server_image_product_code = "SW.VSVR.OS.LNX64.CNTOS.0703.B050"
}
```

Write
Terraform config file (*.tf)
Read
CLI(command line tool)
TERRAFORM CORE
Terraform provider (plugin)
NAVER CLOUD PLATFORM
Provider
Sync
Diff
Terraform state (.tfstate)

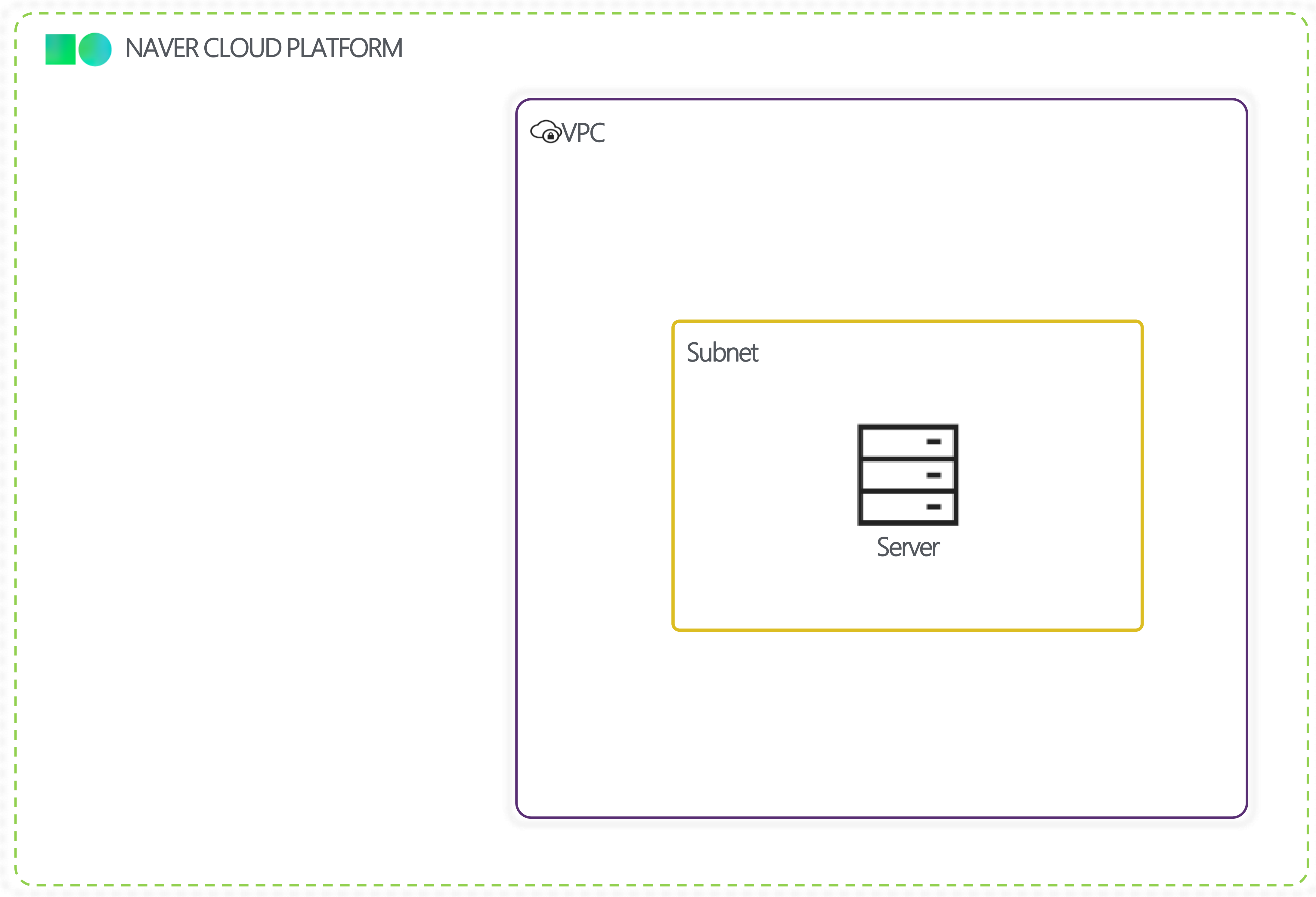
How terraform works

Terraform 으로 VPC 서버 생성을 해봅시다

Terraform config (*.tf)

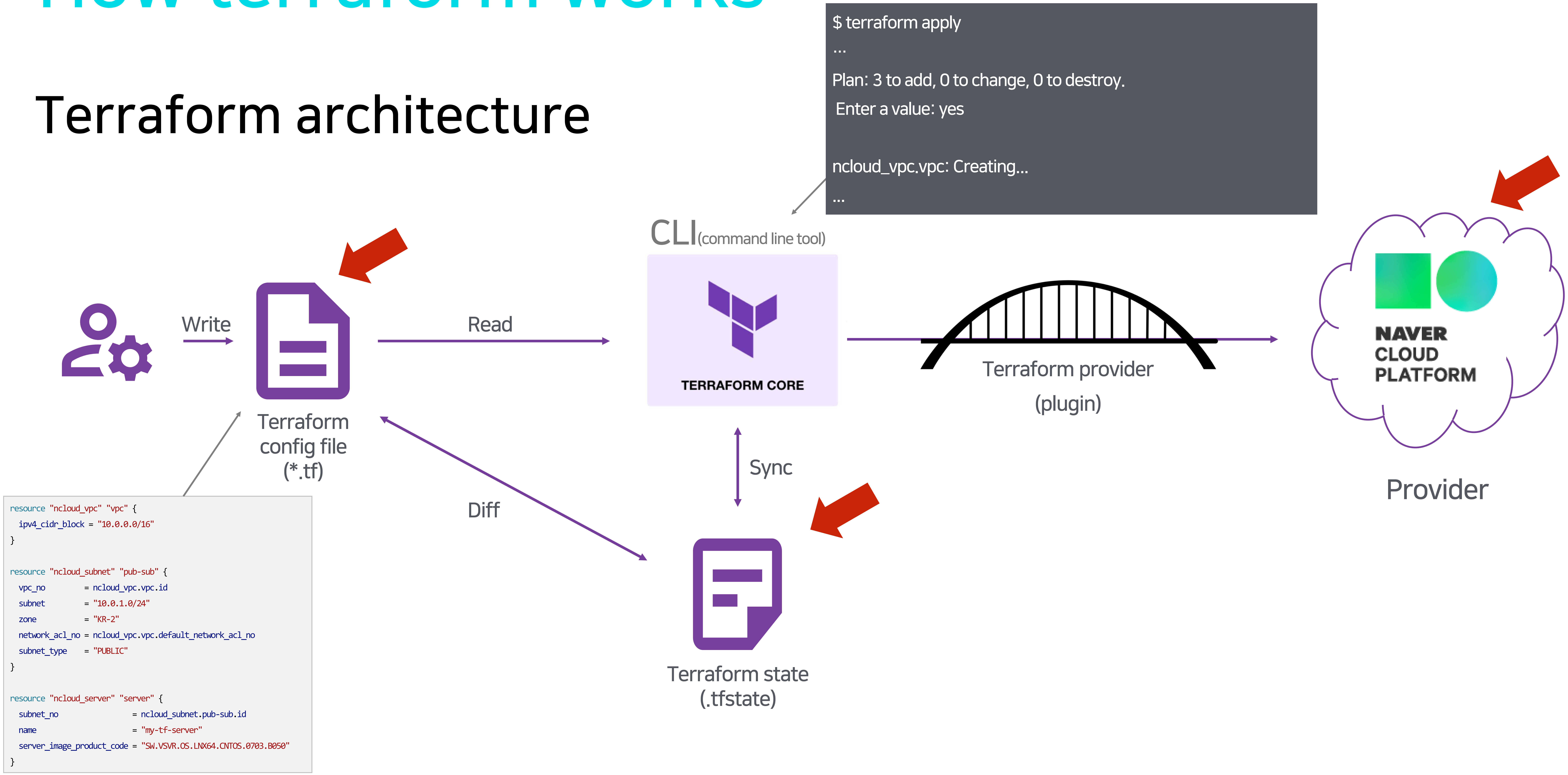
```
resource "ncloud_vpc" "vpc" {
  ipv4_cidr_block = "10.0.0.0/16"
}

resource "ncloud_subnet" "pub-sub" {
  vpc_no         = ncloud_vpc.vpc.id
  subnet         = "10.0.1.0/24"
  zone           = "KR-2"
  network_acl_no = ncloud_vpc.vpc.default_network_acl_no
  subnet_type    = "PUBLIC"
}

resource "ncloud_server" "server" {
  subnet_no                 = ncloud_subnet.pub-sub.id
  name                      = "my-tf-server"
  server_image_product_code = "SW.VSVR.OS.LNX64.CNTOS.0703.B050"
}
```

Provider (TO-BE)



How terraform works

Terraform architecture



```
$ terraform apply
...
Plan: 3 to add, 0 to change, 0 to destroy.
 Enter a value: yes

ncloud_vpc.vpc: Creating...
...
```

```
resource "ncloud_vpc" "vpc" {
  ipv4_cidr_block = "10.0.0.0/16"
}

resource "ncloud_subnet" "pub-sub" {
  vpc_no         = ncloud_vpc.vpc.id
  subnet         = "10.0.1.0/24"
  zone           = "KR-2"
  network_acl_no = ncloud_vpc.vpc.default_network_acl_no
  subnet_type    = "PUBLIC"
}

resource "ncloud_server" "server" {
  subnet_no                 = ncloud_subnet.pub-sub.id
  name                      = "my-tf-server"
  server_image_product_code = "SW.VSVR.OS.LNX64.CNTOS.0703.B050"
}
```

Write
Terraform config file (*.tf)
Read
CLI(command line tool)
TERRAFORM CORE
Terraform provider (plugin)
NAVER CLOUD PLATFORM
Provider
Sync
Diff
Terraform state (.tfstate)

How terraform works

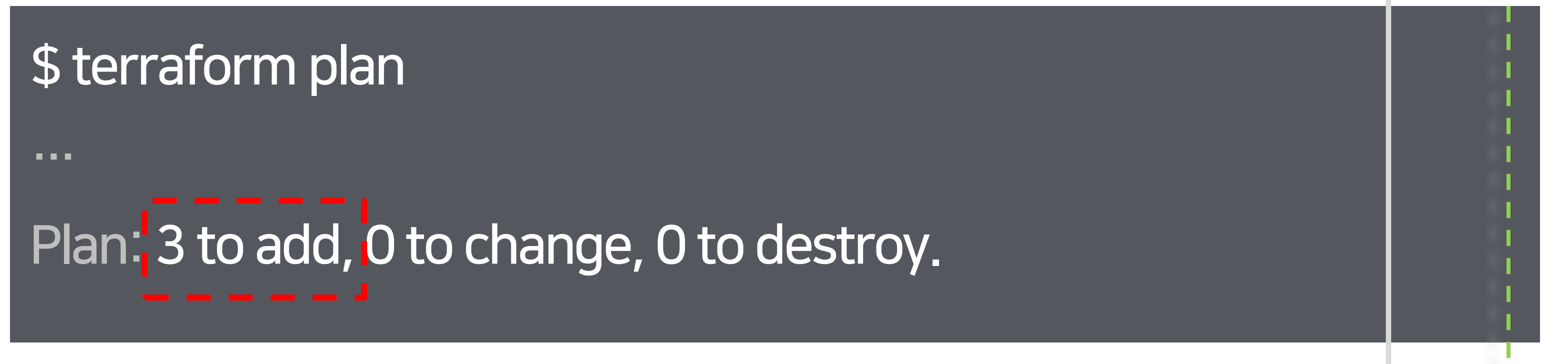
예1) 최초 리소스 생성 시

Terraform config (*.tf)

```
resource "ncloud_vpc" "vpc" {
  ipv4_cidr_block = "10.0.0.0/16"
}

resource "ncloud_subnet" "pub-sub" {
  vpc_no         = ncloud_vpc.vpc.id
  subnet         = "10.0.1.0/24"
  zone           = "KR-2"
  network_acl_no = ncloud_vpc.vpc.default_network_acl_no
  subnet_type    = "PUBLIC"
}

resource "ncloud_server" "server" {
  subnet_no                 = ncloud_subnet.pub-sub.id
  name                      = "my-tf-server"
  server_image_product_code = "SW.VSVR.OS.LNX64.CNTOS.0703.B050"
}
```

plan

Terraform state (.tfstate)



```
$ terraform plan
...
Plan: 3 to add, 0 to change, 0 to destroy.
```

Provider

NAVER CLOUD PLATFORM

How terraform works

예1) 최초 리소스 생성 시

Terraform config (*.tf)

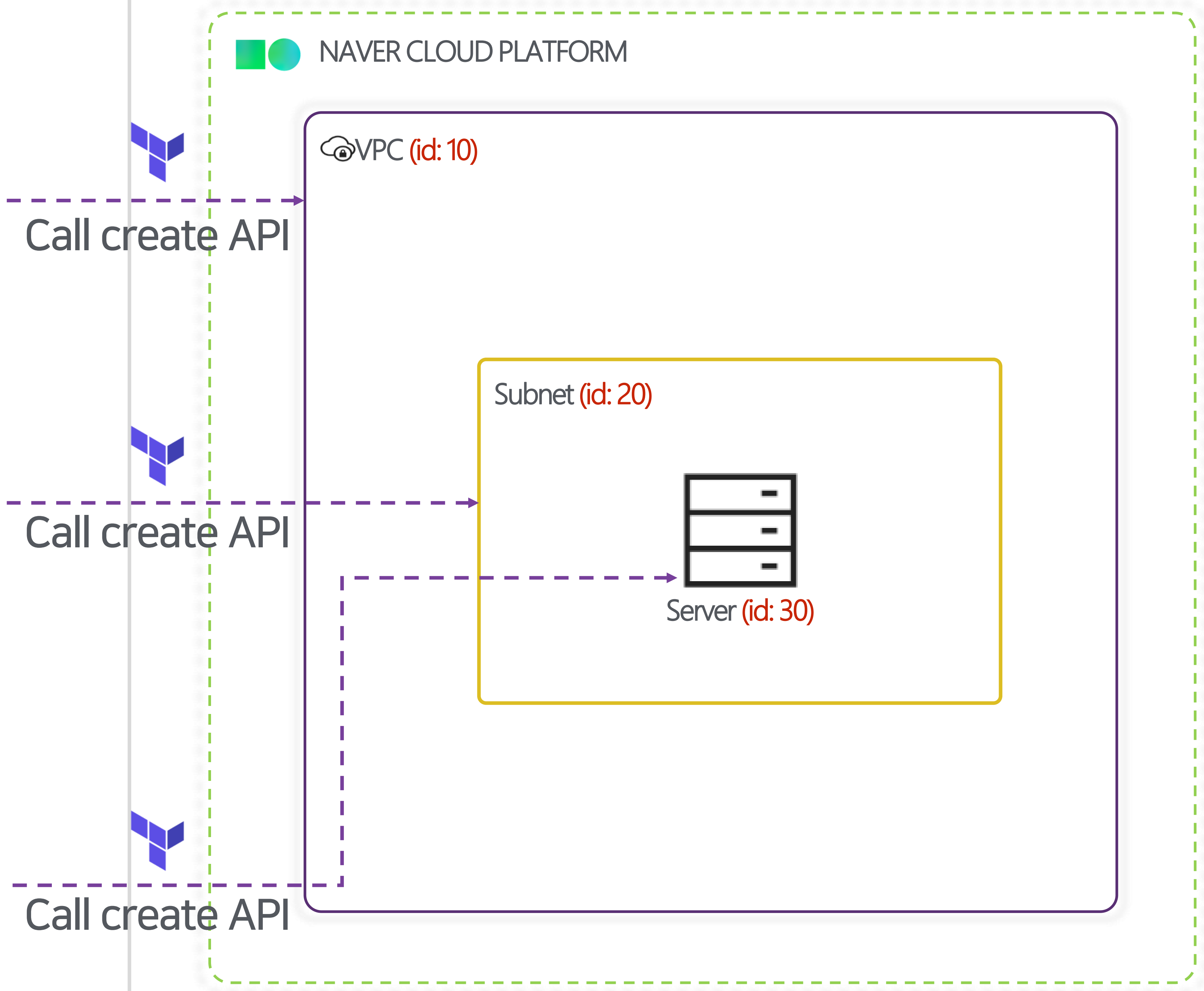
Terraform state (.tfstate)

Provider

```
$ terraform apply
...
Plan: 3 to add, 0 to change, 0 to destroy.
 Enter a value: yes

ncloud_vpc.vpc: Creating...
ncloud_vpc.vpc : Creation complete after 10 [id=10]
ncloud_subnet.pub-sub: Creating...
ncloud_subnet.pub-sub: Still creating... [10s elapsed]
ncloud_subnet.pub-sub: Creation complete after 13s [id=20]
ncloud_server.server: Creating...
...
ncloud_server.server: Creation complete after 59s [id=30]
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```

```
re
}

re

}

re
  name                      = "my-tf-server"
  server_image_product_code = "SW.VSVR.OS.LNX64.CNTOS.0703.B050"
}
```

✓ apply



How terraform works

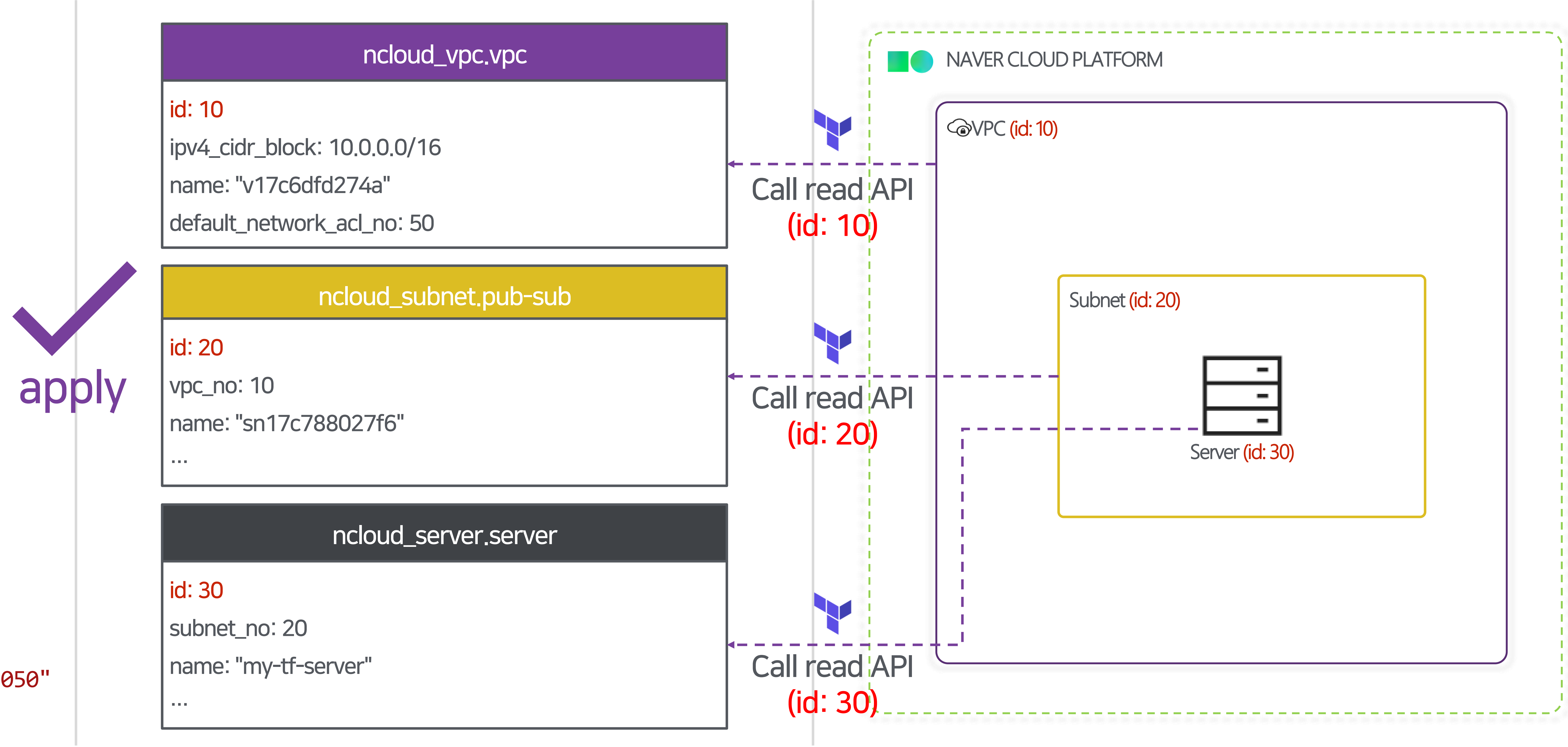
예1) 최초 리소스 생성 시

Terraform config (*.tf)

```
resource "ncloud_vpc" "vpc" {
  ipv4_cidr_block = "10.0.0.0/16"
}

resource "ncloud_subnet" "pub-sub" {
  vpc_no         = ncloud_vpc.vpc.id
  subnet         = "10.0.1.0/24"
  zone           = "KR-2"
  network_acl_no = ncloud_vpc.vpc.default_network_acl_no
  subnet_type    = "PUBLIC"
}

resource "ncloud_server" "server" {
  subnet_no                 = ncloud_subnet.pub-sub.id
  name                      = "my-tf-server"
  server_image_product_code = "SW.VSVR.OS.LNX64.CNTOS.0703.B050"
}
```

Terraform state (.tfstate)

Provider



ncloud_vpc.vpc

id: 10
ipv4_cidr_block: 10.0.0.0/16
name: "v17c6dfd274a"
default_network_acl_no: 50

ncloud_subnet.pub-sub

id: 20
vpc_no: 10
name: "sn17c788027f6"
...

ncloud_server.server

id: 30
subnet_no: 20
name: "my-tf-server"
...

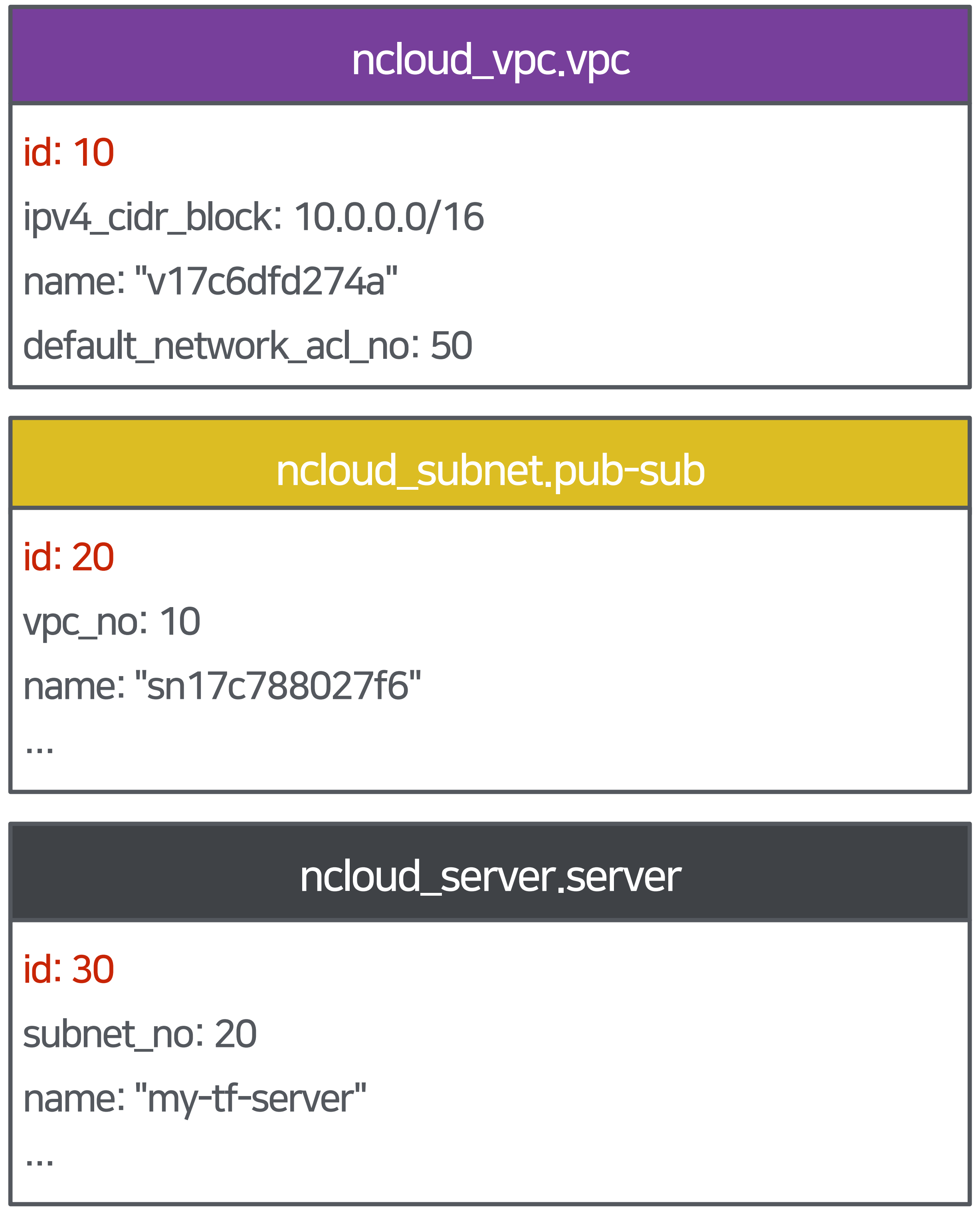
How terraform works

예2) 코드에서 리소스를 삭제 한 경우

Terraform config (*.tf)

```
resource "ncloud_vpc" "vpc" {
  ipv4_cidr_block = "10.0.0.0/16"
}

resource "ncloud_subnet" "pub-sub" {
  vpc_no         = ncloud_vpc.vpc.id
  subnet         = "10.0.1.0/24"
  zone           = "KR-2"
  network_acl_no = ncloud_vpc.vpc.default_network_acl_no
  subnet_type    = "PUBLIC"
}

# resource "ncloud_server" "server" {
#  subnet_no                 = ncloud_subnet.pub-sub.id
#  name                      = "my-tf-server"
#  server_image_product_code = "SW.VSVR.OS.LNX64.CNTOS.0703.B050"
# }
```

Terraform state (.tfstate)



ncloud_vpc.vpc

id: 10
ipv4_cidr_block: 10.0.0.0/16
name: "v17c6dfd274a"
default_network_acl_no: 50

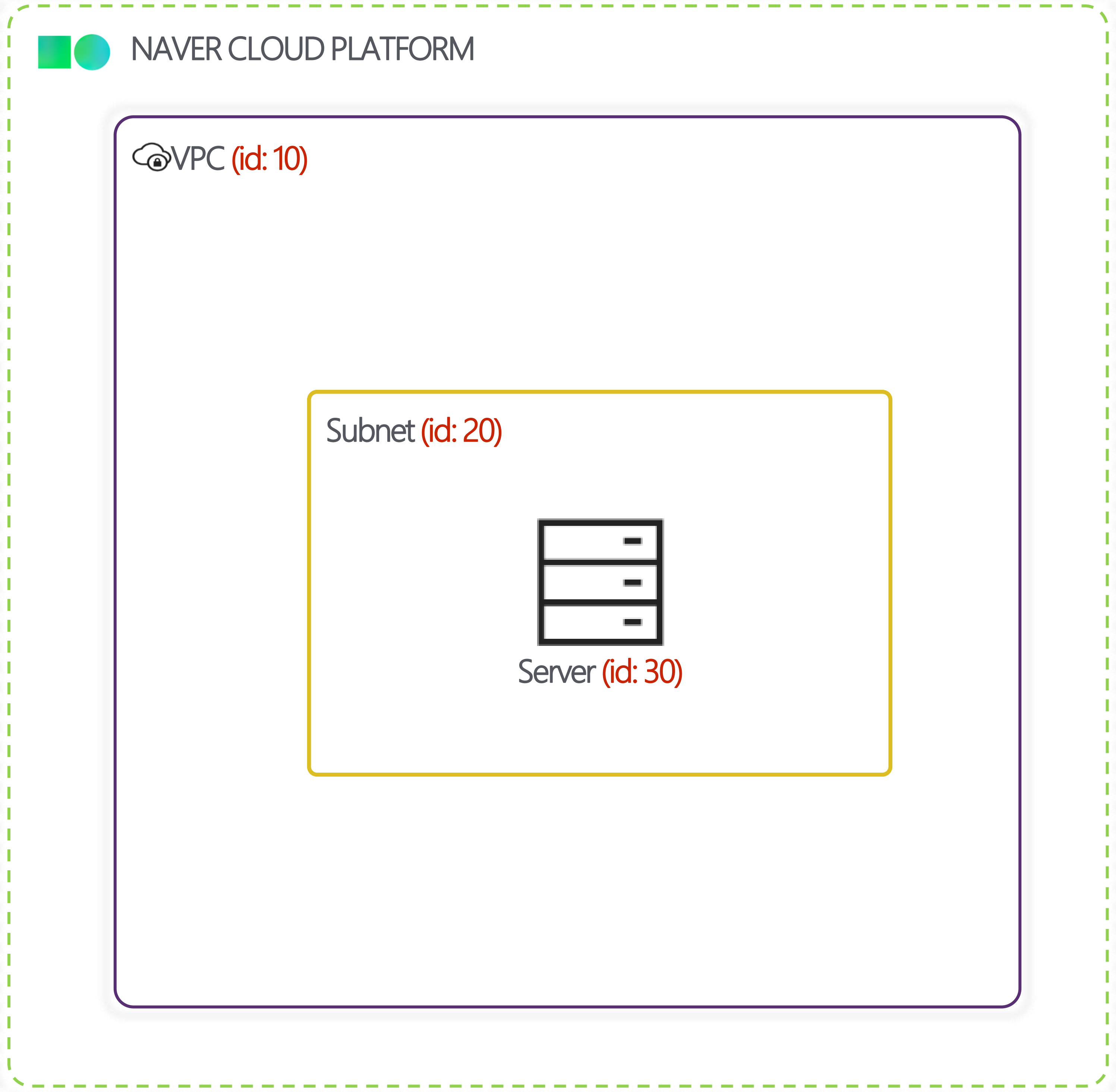
ncloud_subnet.pub-sub

id: 20
vpc_no: 10
name: "sn17c788027f6"
...

ncloud_server.server

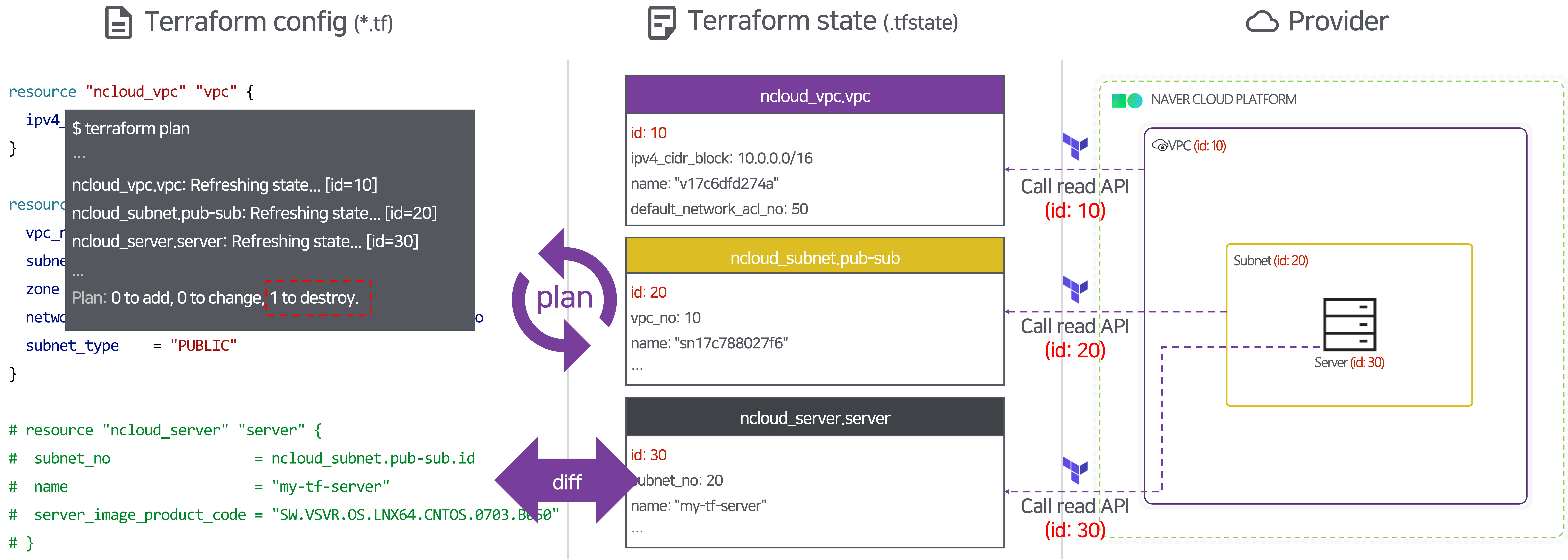
id: 30
subnet_no: 20
name: "my-tf-server"
...

Provider



How terraform works

예2) 코드에서 리소스를 삭제 한 경우



Terraform config (*.tf)

```
resource "ncloud_vpc" "vpc" {
  ipv4_
}

resourc
  vpc_n
  subne
  zone
  netwo                                              o
  subnet_type      = "PUBLIC"
}

# resource "ncloud_server" "server" {
#   subnet_no                 = ncloud_subnet.pub-sub.id
#   name                      = "my-tf-server"
#   server_image_product_code = "SW.VSVR.OS.LNX64.CNTOS.0703.B050"
# }
```

```
$ terraform plan
...
ncloud_vpc.vpc: Refreshing state... [id=10]
ncloud_subnet.pub-sub: Refreshing state... [id=20]
ncloud_server.server: Refreshing state... [id=30]
...
Plan: 0 to add, 0 to change, 1 to destroy.
```

Terraform state (.tfstate)

ncloud_vpc.vpc

id: 10
ipv4_cidr_block: 10.0.0.0/16
name: "v17c6dfd274a"
default_network_acl_no: 50

ncloud_subnet.pub-sub

id: 20
vpc_no: 10
name: "sn17c788027f6"
...

ncloud_server.server

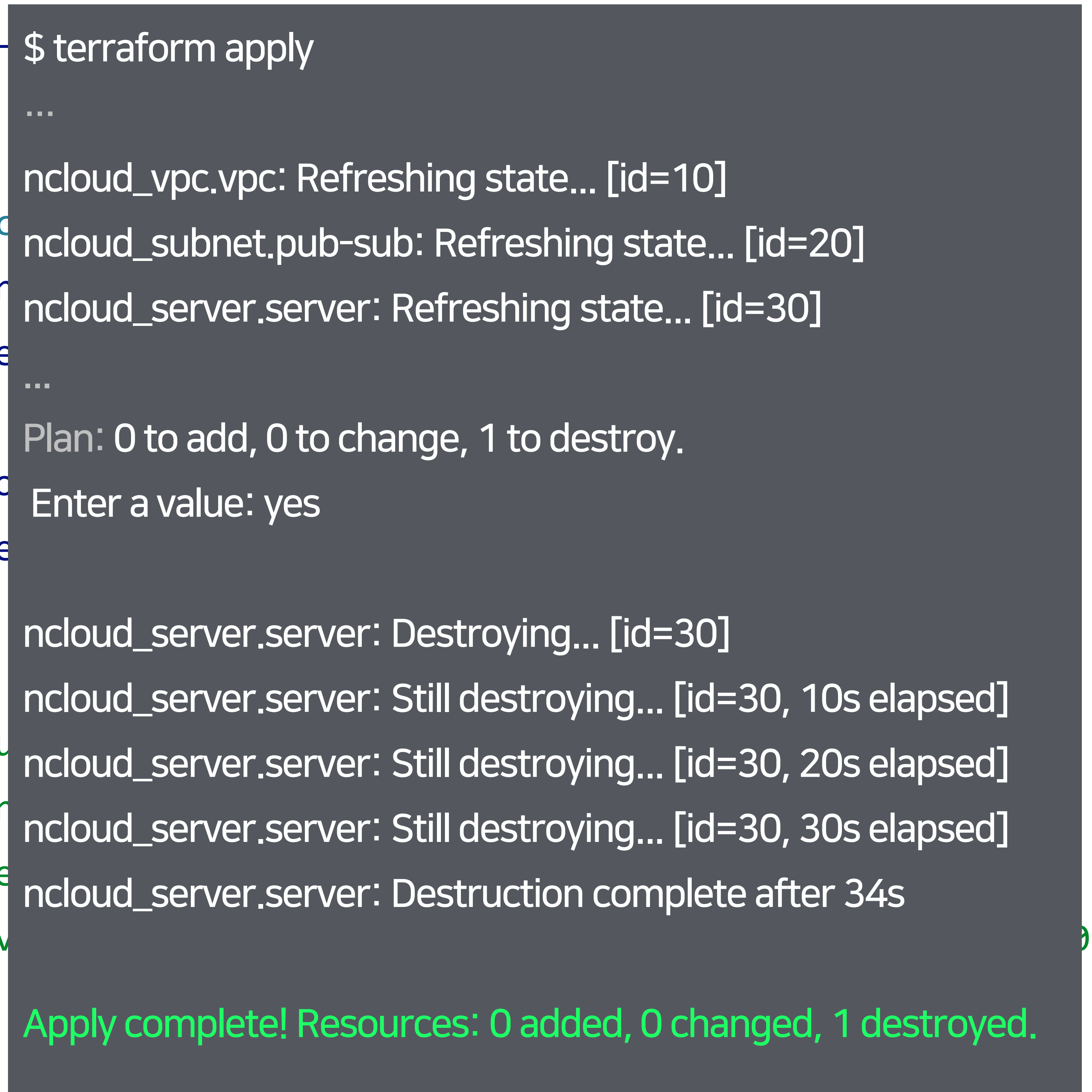
id: 30
subnet_no: 20
name: "my-tf-server"
...

How terraform works

예2) 코드에서 리소스를 삭제 한 경우

Terraform config (*.tf)

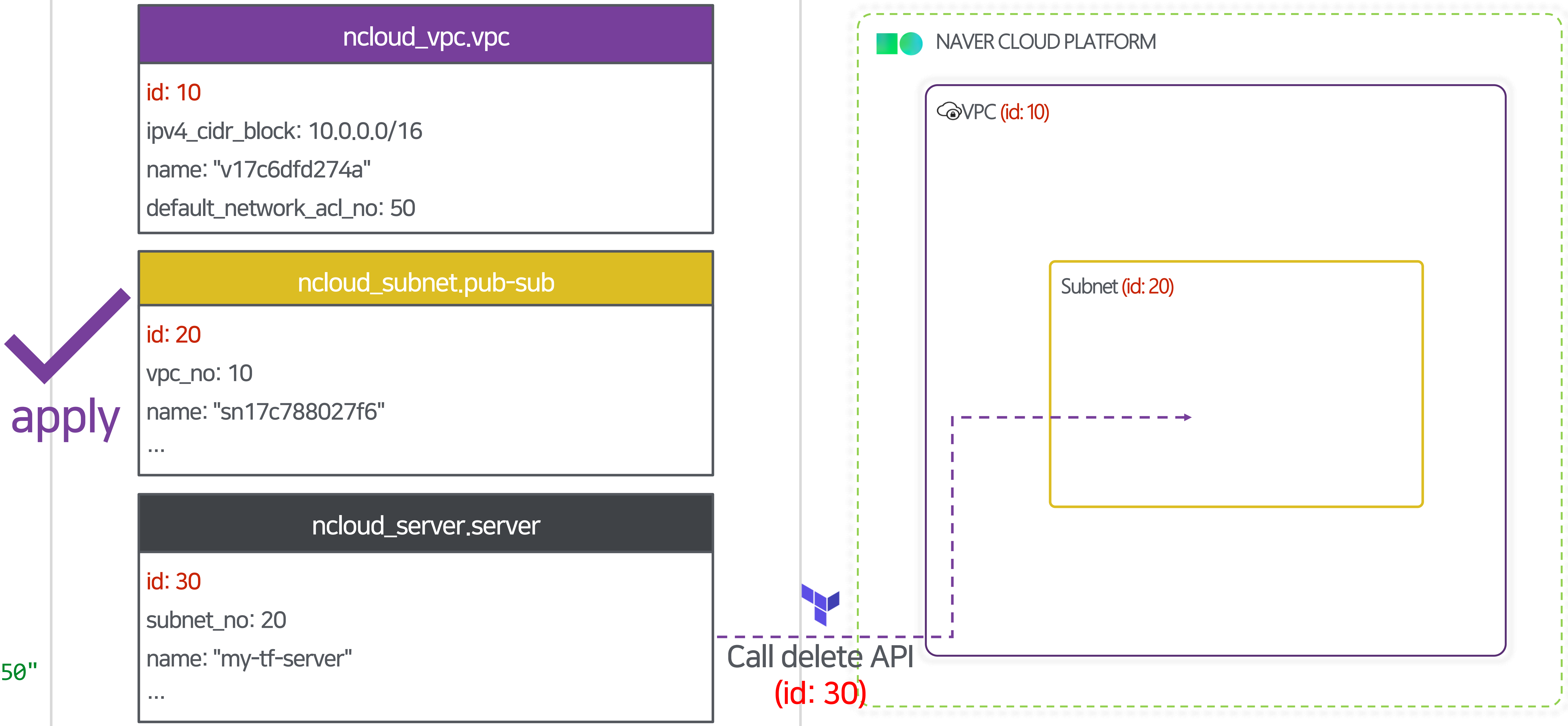
```
resource "ncloud_vpc" "vpc" {
  ipv4_
}

resourc
  vpc_r
  subne
  zone
  netwo
  subne
}

# resou
#  subr
#  name
#  serv                                                   050"
# }
```



```
$ terraform apply
...
ncloud_vpc.vpc: Refreshing state... [id=10]
ncloud_subnet.pub-sub: Refreshing state... [id=20]
ncloud_server.server: Refreshing state... [id=30]
...
Plan: 0 to add, 0 to change, 1 to destroy.
 Enter a value: yes

ncloud_server.server: Destroying... [id=30]
ncloud_server.server: Still destroying... [id=30, 10s elapsed]
ncloud_server.server: Still destroying... [id=30, 20s elapsed]
ncloud_server.server: Still destroying... [id=30, 30s elapsed]
ncloud_server.server: Destruction complete after 34s

Apply complete! Resources: 0 added, 0 changed, 1 destroyed.
```

Terraform state (.tfstate)

Provider



How terraform works

예2) 코드에서 리소스를 삭제 한 경우

Terraform config (*.tf)

```
resource "ncloud_vpc" "vpc" {
  ipv4_cidr_block = "10.0.0.0/16"
}

resource "ncloud_subnet" "pub-sub" {
  vpc_no         = ncloud_vpc.vpc.id
  subnet         = "10.0.1.0/24"
  zone           = "KR-2"
  network_acl_no = ncloud_vpc.vpc.default_network_acl_no
  subnet_type    = "PUBLIC"
}

# resource "ncloud_server" "server" {
#   subnet_no                 = ncloud_subnet.pub-sub.id
#   name                      = "my-tf-server"
#   server_image_product_code = "SW.VSVR.OS.LNX64.CNTOS.0703.B050"
# }
```

Terraform state (.tfstate)

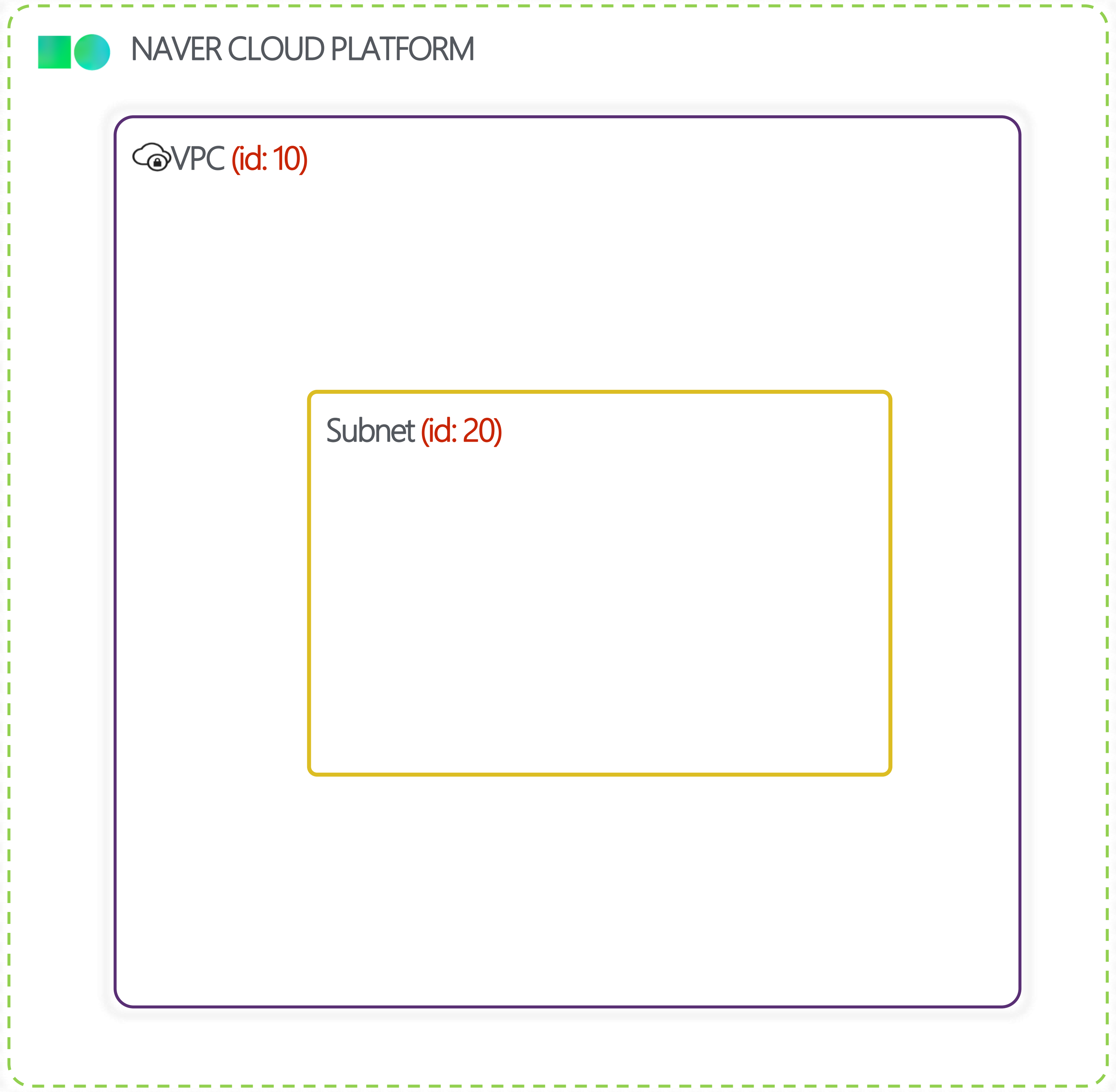
ncloud_vpc.vpc

id: 10
ipv4_cidr_block: 10.0.0.0/16
name: "v17c6dfd274a"
default_network_acl_no: 50

ncloud_subnet.pub-sub

id: 20
vpc_no: 10
name: "sn17c788027f6"
...

Provider



How terraform works

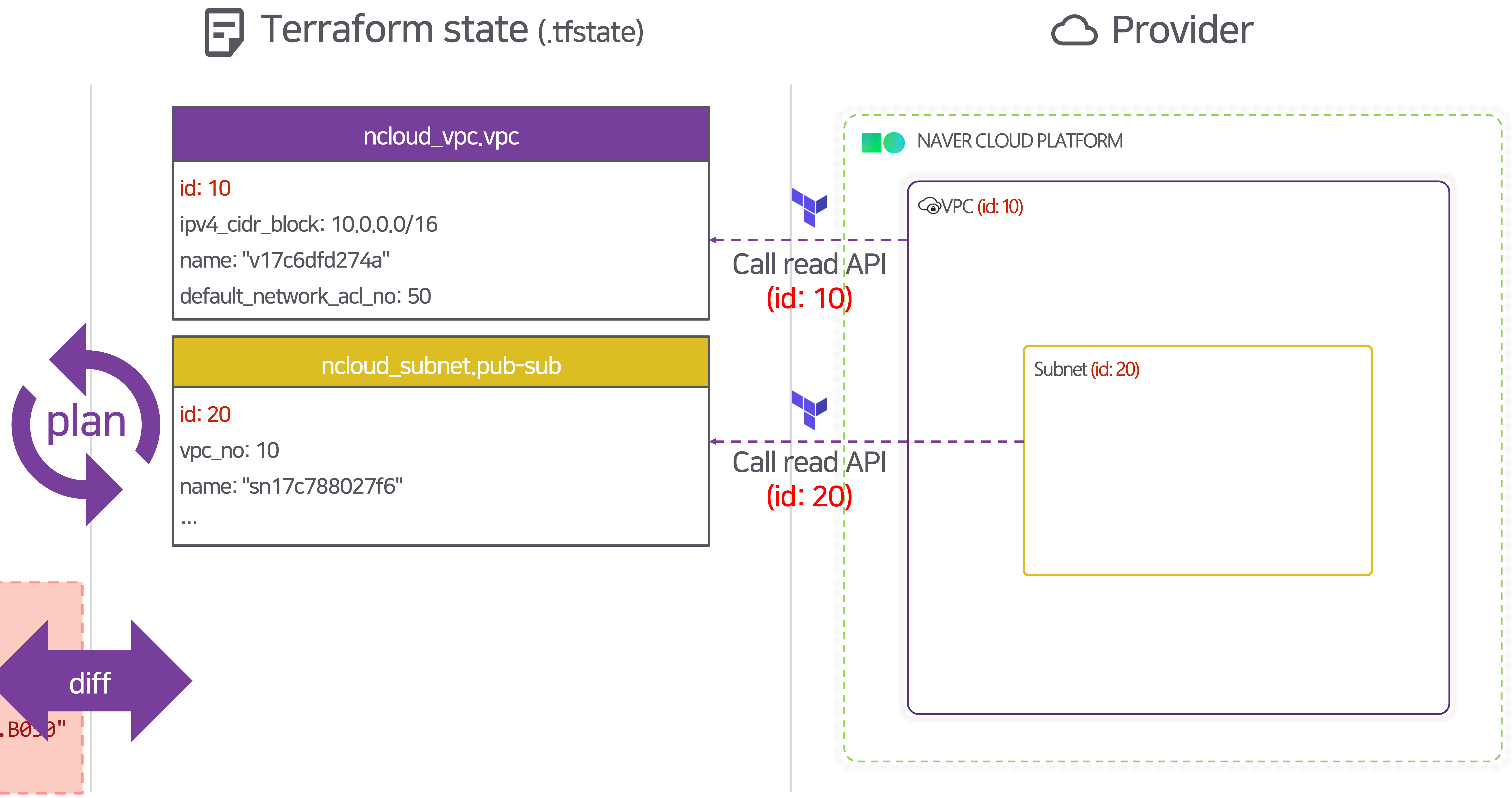
예3) 코드에서 리소스를 추가 한 경우

Terraform config (*.tf)

```
resource "ncloud_vpc" "vpc" {
  ipv4_
}

resourc
  vpc_n
  subne
  zone               = "KR-2"
  network_acl_no = ncloud_vpc.vpc.default_network_acl_no
  subnet_type      = "PUBLIC"
}
```

```
$ terraform plan
...
ncloud_vpc.vpc: Refreshing state... [id=10]
ncloud_subnet.pub-sub: Refreshing state... [id=20]
...
Plan: 1 to add, 0 to change, 0 to destroy.
```

```
resource "ncloud_server" "server" {
  subnet_no                  = ncloud_subnet.pub-sub.id
  name                       = "my-tf-server"
  server_image_product_code = "SW.VSVR.OS.LNX64.CNTOS.0703.B050"
}
```



Terraform state (.tfstate)

ncloud_vpc.vpc

id: 10
ipv4_cidr_block: 10.0.0.0/16
name: "v17c6dfd274a"
default_network_acl_no: 50

ncloud_subnet.pub-sub

id: 20
vpc_no: 10
name: "sn17c788027f6"
...

How terraform works

예3) 코드에서 리소스를 추가 한 경우

Terraform config (*.tf)

```
resource "ncloud_vpc" "vpc" {
  ipv4_
}

resourc
  vpc_r
  subne
  zone
  netwo
  subne
}

resourc
  subne
  name
  serve                                   50"
}
```

```
$ terraform apply
...
ncloud_vpc.vpc: Refreshing state... [id=10]
ncloud_subnet.pub-sub: Refreshing state... [id=20]
...
Plan: 1 to add, 0 to change, 0 to destroy.
 Enter a value: yes

ncloud_server.server: Creating...
ncloud_server.server: Still creating... [10s elapsed]
ncloud_server.server: Still creating... [20s elapsed]
ncloud_server.server: Still creating... [30s elapsed]
ncloud_server.server: Creation complete after 34s [id=31]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

✓ apply

Terraform state (.tfstate)

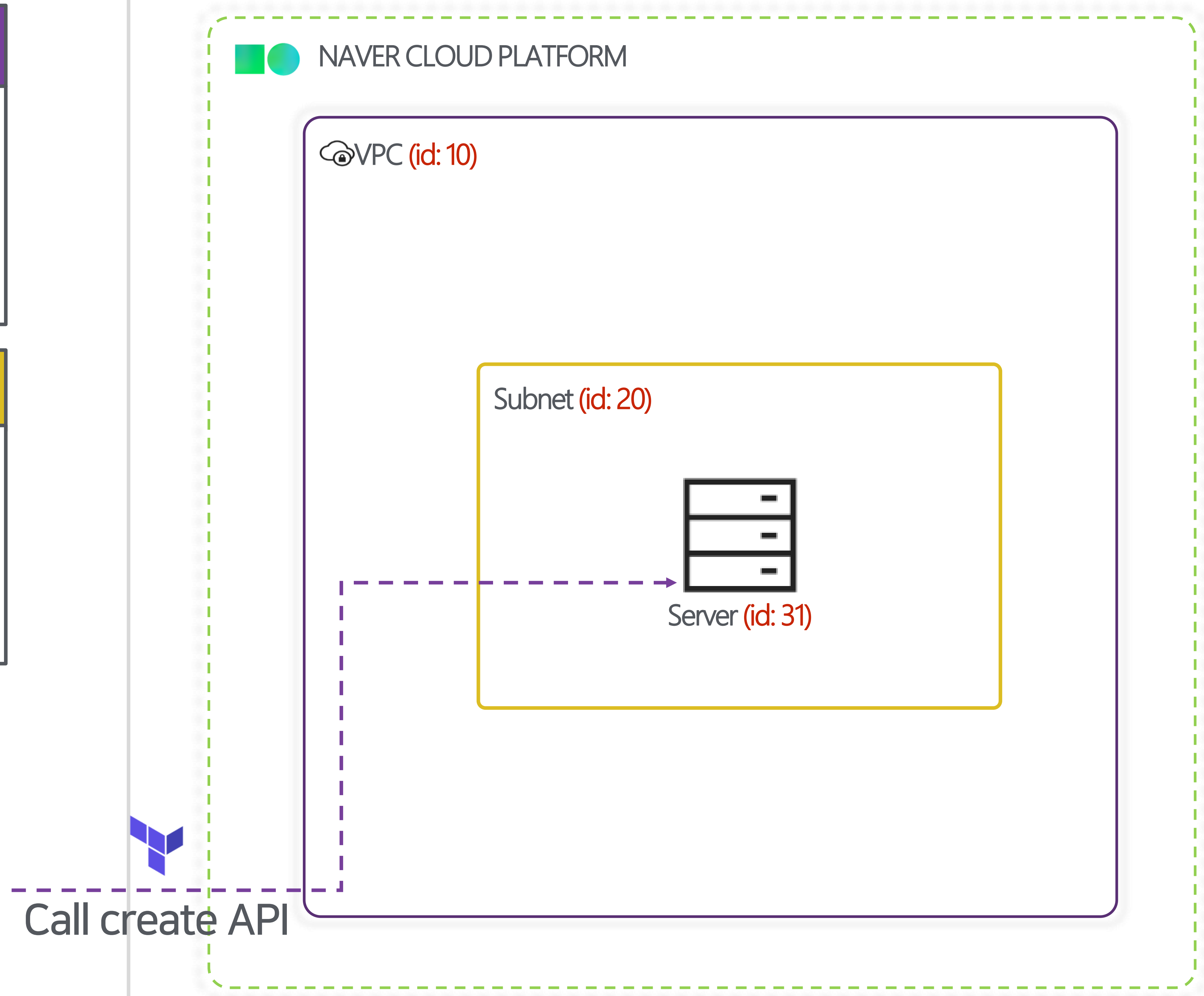
ncloud_vpc.vpc

id: 10
ipv4_cidr_block: 10.0.0.0/16
name: "v17c6dfd274a"
default_network_acl_no: 50

ncloud_subnet.pub-sub

id: 20
vpc_no: 10
name: "sn17c788027f6"
...

Provider



How terraform works

예3) 코드에서 리소스를 추가 한 경우

Terraform config (*.tf)

```
resource "ncloud_vpc" "vpc" {
  ipv4_
}

resourc
  vpc_r
  subne
  zone
  netwo
  subne
}

resourc
  subne
  name
  serve                                                          50"
}
```

```
$ terraform apply
...
ncloud_vpc.vpc: Refreshing state... [id=10]
ncloud_subnet.pub-sub: Refreshing state... [id=20]
...
Plan: 1 to add, 0 to change, 0 to destroy.
 Enter a value: yes

ncloud_server.server: Creating...
ncloud_server.server: Still creating... [10s elapsed]
ncloud_server.server: Still creating... [20s elapsed]
ncloud_server.server: Still creating... [30s elapsed]
ncloud_server.server: Creation complete after 34s [id=31]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

✓ apply

Terraform state (.tfstate)

ncloud_vpc.vpc

id: 10
ipv4_cidr_block: 10.0.0.0/16
name: "v17c6dfd274a"
default_network_acl_no: 50

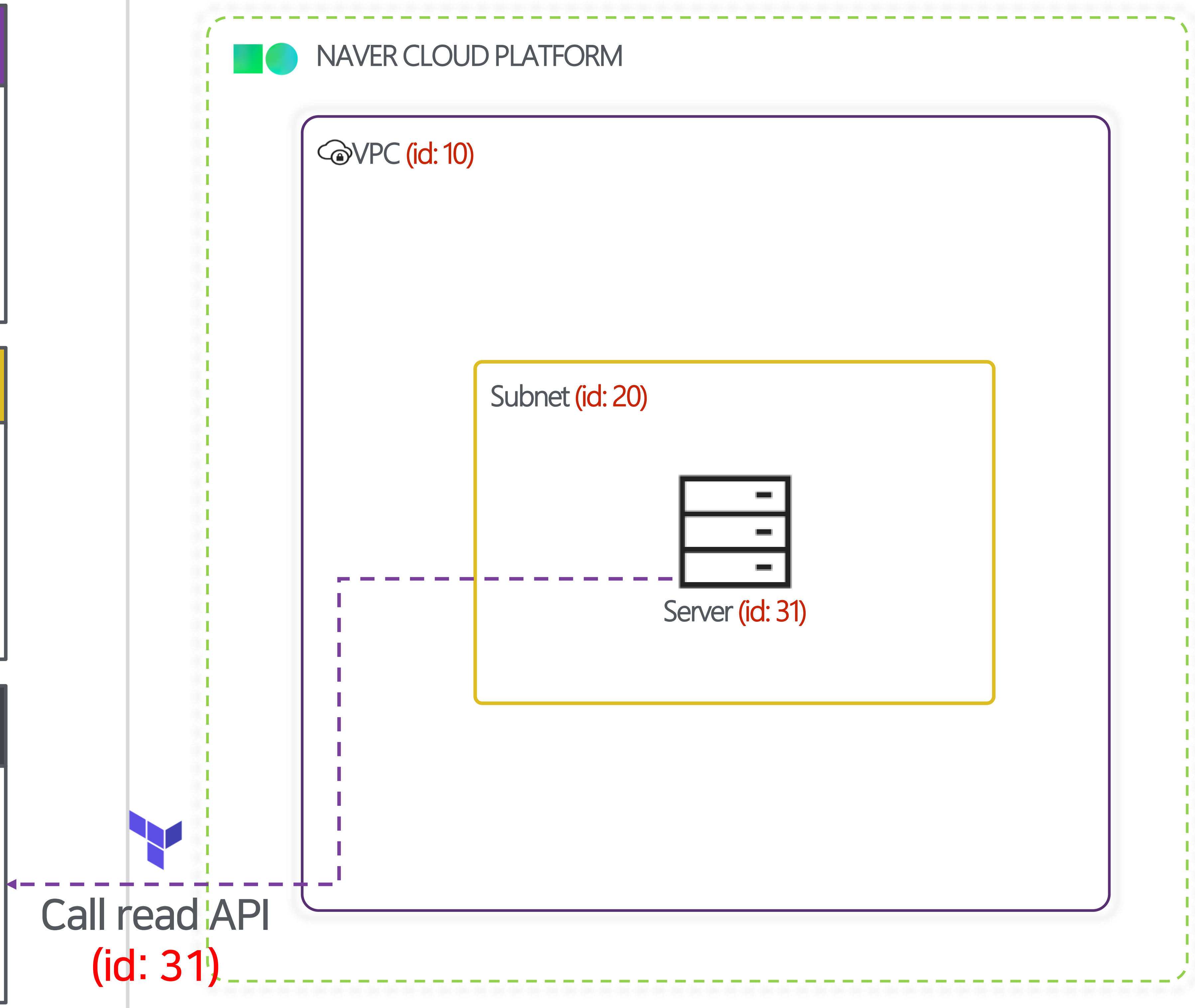
ncloud_subnet.pub-sub

id: 20
vpc_no: 10
name: "sn17c788027f6"
...

ncloud_server.server

id: 31
subnet_no: 20
name: "my-tf-server"
...

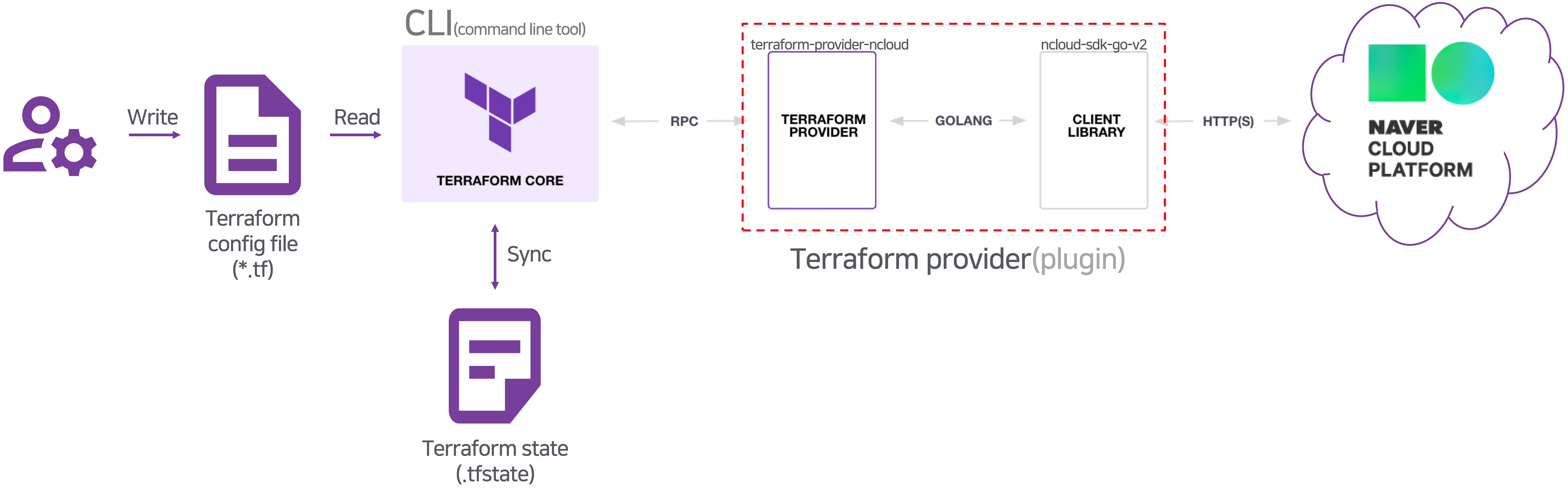
Provider



How terraform works

Terraform은 어떻게 동작할까?

Terraform provider(plugin)의 CRUD operation을 통해,
인프라를 반영하고 상태를 업데이트

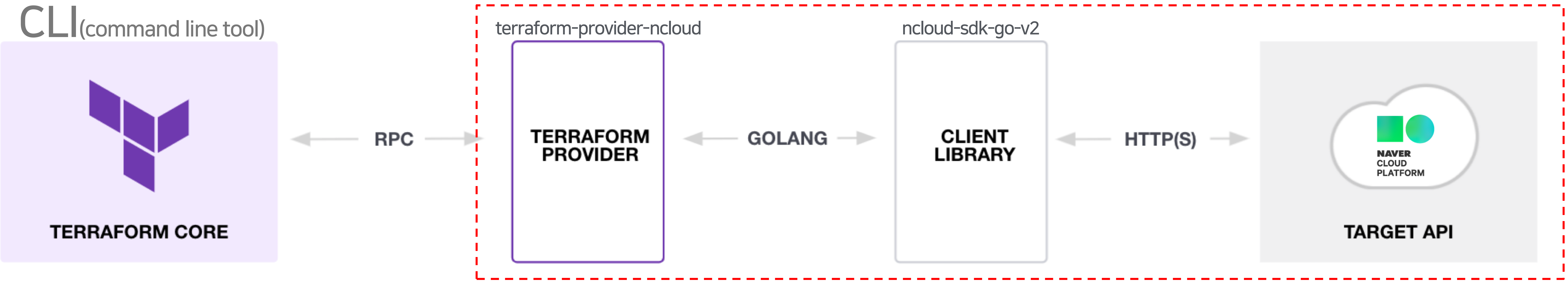


3. Terraform provider 만들기

Terraform provider 만들기

Terraform provider를 만들기 위해 필요한 것들

1. CRUD를 지원하는 REST API (NCLOUD API)
2. REST API 를 호출하는 GO CLIENT LIBRARY (NCLOUD GO SDK)
3. Terraform plugin

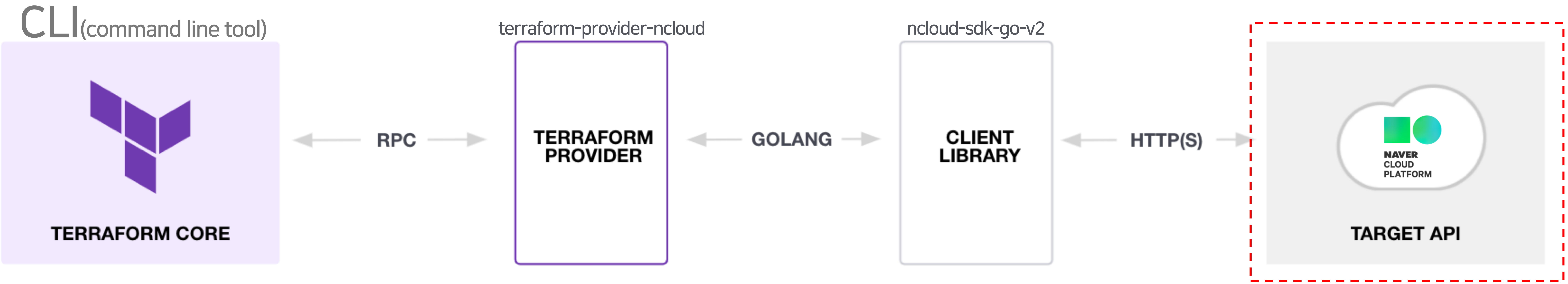


https://github.com/NaverCloudPlatform/terraform-provider-ncloud

Terraform provider 만들기

Terraform provider를 만들기 위해 필요한 것들

1. CRUD를 지원하는 REST API (NCLOUD API)
2. REST API 를 호출하는 GO CLIENT LIBRARY (NCLOUD GO SDK)
3. Terraform plugin

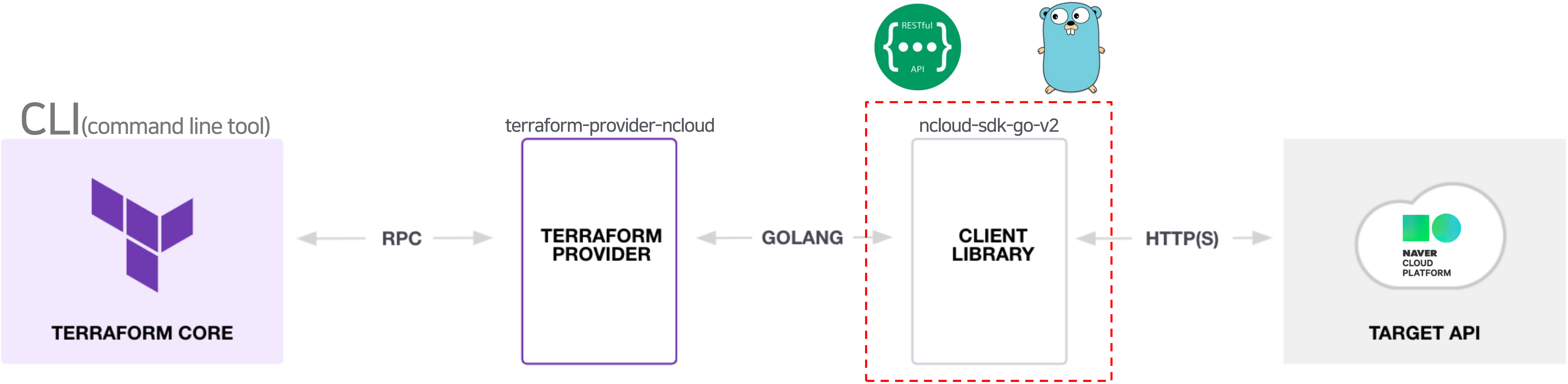


https://github.com/NaverCloudPlatform/ncloud-sdk-go-v2

NCLOUD GO SDK

Terraform provider를 만들기 위해 필요한 것들

1. CRUD를 지원하는 REST API (NCLOUD API)
2. REST API 를 호출하는 GO CLIENT LIBRARY (NCLOUD GO SDK)
3. Terraform plugin

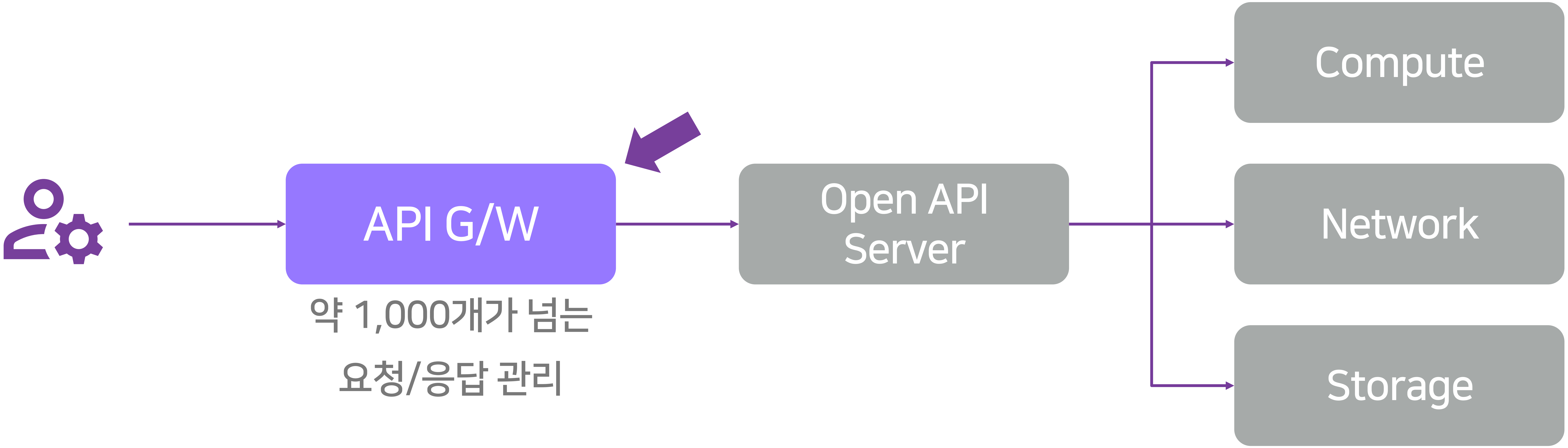


https://github.com/NaverCloudPlatform/ncloud-sdk-go-v2

NCLOUD GO SDK

Go SDK 를 자동으로 만들자

- API G/W 에서 1,000개 이상의 요청/응답 메타 정보 관리



https://github.com/NaverCloudPlatform/ncloud-sdk-go-v2

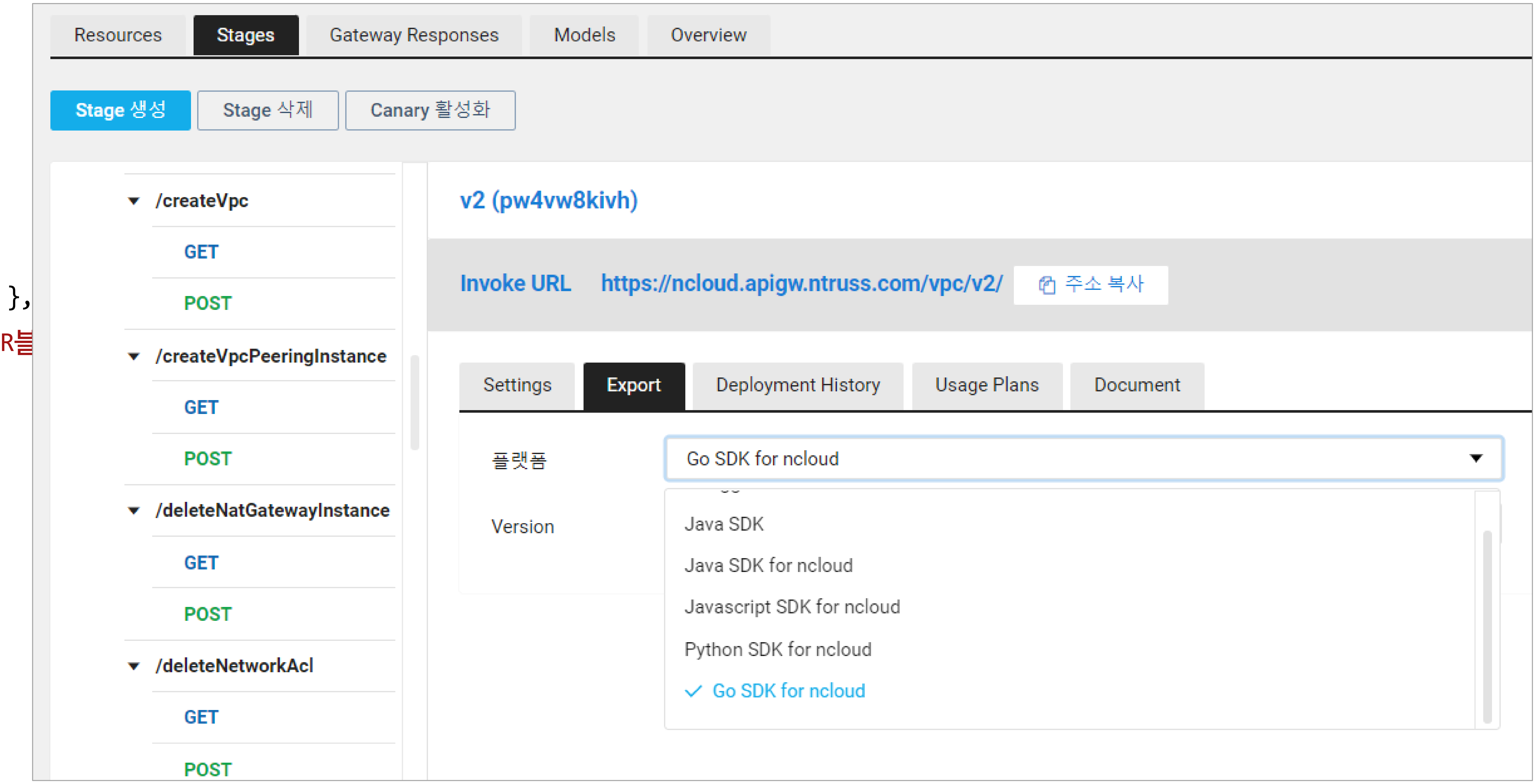
NCLOUD GO SDK

Go SDK 를 자동으로 만들자

- Swagger meta data를 이용한 SDK 추출

CreateVpcRequest

```
{
  "type": "object",
  "required": ["ipv4CidrBlock"],
  "properties": {
    "regionCode": { "type": "string", "description": "REGION코드" },
    "ipv4CidrBlock": { "type": "string", "description": "IPv4 CIDR블
    "vpcName": { "type": "string", "description": "VPC이름" },
    "responseFormatType": {
      "type": "string",
      "description": "responseFormatType {json, xml}"
    }
  },
  "title": "createVpcRequest"
}
```



v2 (pw4vw8kivh)

Invoke URL https://ncloud.apigw.ntruss.com/vpc/v2/

플랫폼 Go SDK for ncloud

Version

https://github.com/NaverCloudPlatform/ncloud-sdk-go-v2

NCLOUD GO SDK

Go SDK 를 자동으로 만들자

- Swagger meta data를 이용한 SDK 추출

CreateVpcRequest

```
{
  "type": "object",
  "required": ["ipv4CidrBlock"],
  "properties": {
    "regionCode": { "type": "string", "description": "REGION코드" },
    "ipv4CidrBlock": { "type": "string", "description": "IPv4 CIDR블록" },
    "vpcName": { "type": "string", "description": "VPC이름" },
    "responseFormatType": {
      "type": "string",
      "description": "responseFormatType {json, xml}"
    }
  },
  "title": "createVpcRequest"
}
```

Swagger
Codegen
Build Go SDK

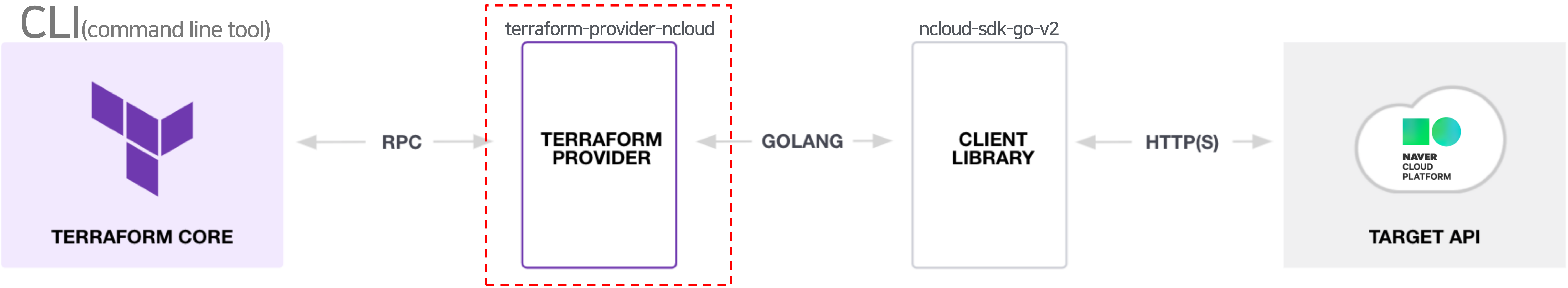
```
/* V2ApiService
 VPC생성
 @param createVpcRequest createVpcRequest
 @return *CreateVpcResponse*/
func (a *V2ApiService) CreateVpc(createVpcRequest
*CreateVpcRequest) (*CreateVpcResponse, error) {
    var (
        localVarHttpMethod = strings.ToUpper("Post")
        localVarPostBody interface{}
        localVarFileName string
        localVarFileBytes []byte
        successPayload  CreateVpcResponse
    )

    // create path and map variables
    localVarPath := a.client.cfg.BasePath + "/createVpc"

    localVarHeaderParams := make(map[string]string)
    localVarQueryParams := url.Values{}
    localVarFormParams := url.Values{}

...
```

https://github.com/NaverCloudPlatform/ncloud-sdk-go-v2

Terraform provider 만들기

Terraform provider를 만들기 위해 필요한 것들

1. CRUD를 지원하는 REST API (NCLOUD API)
2. REST API 를 호출하는 GO CLIENT LIBRARY (NCLOUD GO SDK)
3. Terraform plugin



https://github.com/NaverCloudPlatform/terraform-provider-ncloud

Terraform plugin 개발

Terraform plugin이 해야 할 일

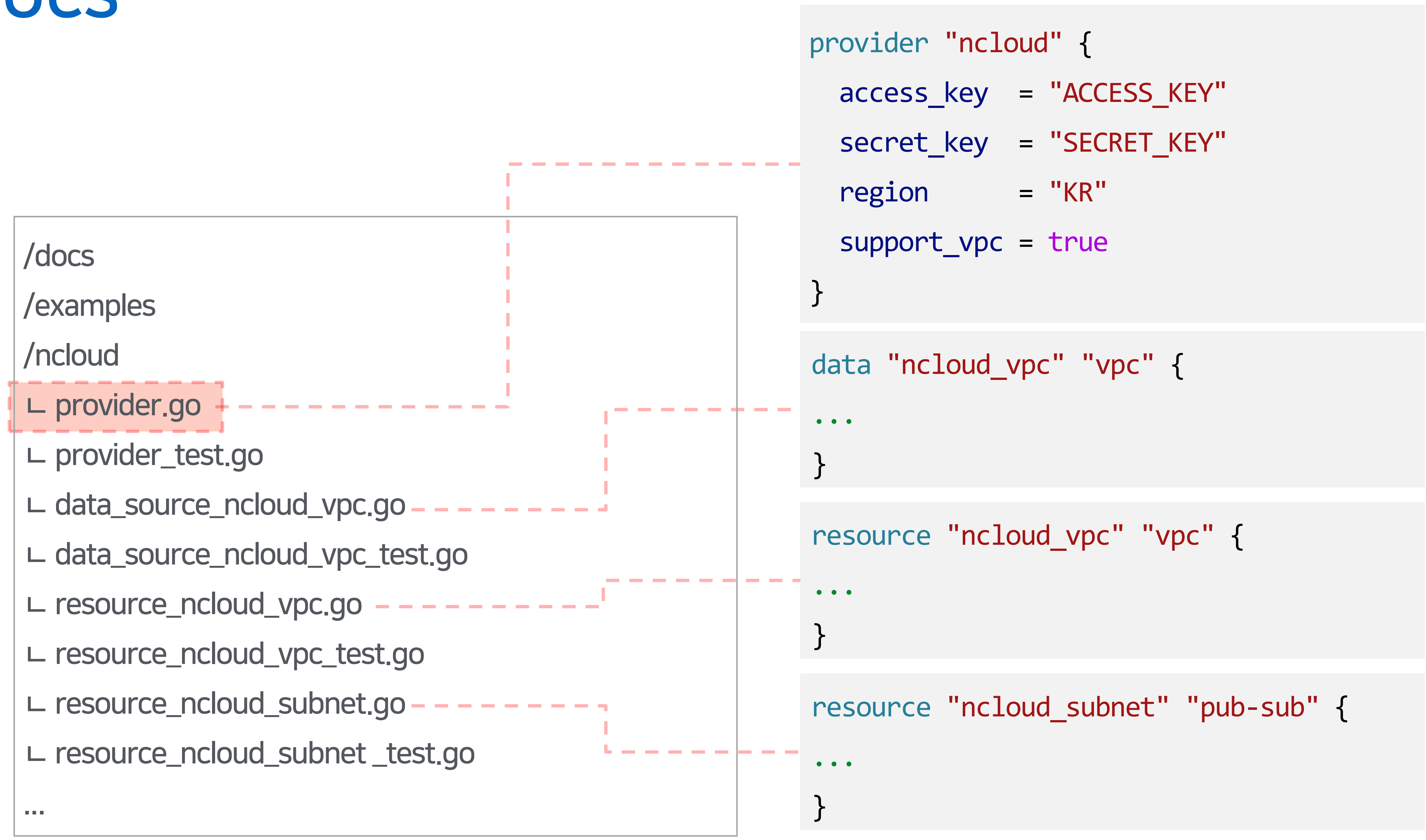
- API 호출에 사용되는 포함된 라이브러리의 **인증** 및 **초기화**
- 서비스에 매핑 되는 **리소스** 정의

```
provider "ncloud" {
  access_key  = "ACCESS_KEY"
  secret_key  = "SECRET_KEY"
  region      = "KR"
}

resource "ncloud_server" "server" {
  subnet_no                 = ncloud_subnet.pub-sub.id
  name                      = "my-tf-server"
  server_image_product_code = "SW.VSVR.OS.LNX64.CNTOS.0703.B050"
}
```

Terraform plugin 개발

Terraform plugin 프로젝트 구조

- Resource와 Datasource는 /ncloud 에 위치
- 예제들은 /example 그리고 문서는 /docs
- Provider정의는 provider.go
- terraform-provider-scaffolding
  을 통해 빠른 시작 가능



```
/docs
/examples
/ncloud
∟ provider.go
∟ provider_test.go
∟ data_source_ncloud_vpc.go
∟ data_source_ncloud_vpc_test.go
∟ resource_ncloud_vpc.go
∟ resource_ncloud_vpc_test.go
∟ resource_ncloud_subnet.go
∟ resource_ncloud_subnet _test.go
...
```

```
provider "ncloud" {
  access_key  = "ACCESS_KEY"
  secret_key  = "SECRET_KEY"
  region      = "KR"
  support_vpc = true
}
```

```
data "ncloud_vpc" "vpc" {
...
}
```

```
resource "ncloud_vpc" "vpc" {
...
}
```

```
resource "ncloud_subnet" "pub-sub" {
...
}
```

https://github.com/NaverCloudPlatform/terraform-provider-ncloud

Provider를 정의하자

Schema를 정의 하자

인증키, 리젼 등 메타 정보를 정의

provider.go

```go
func Provider() *schema.Provider {
    return &schema.Provider{
        Schema:         schemaMap(),
        DataSourcesMap: map[string]*schema.Resource{
            "ncloud_vpc": dataSourceNcloudVpc(),
            "ncloud_subnet": dataSourceNcloudSubnet(),
            // ...
        },
        ResourcesMap: map[string]*schema.Resource{
            "ncloud_vpc": resourceNcloudVpc(),
            "ncloud_subnet": resourceNcloudSubnet(),
            // ...
        },
        ConfigureFunc:  providerConfigure,
    }
}
```

```go
func schemaMap() map[string]*schema.Schema {
    return map[string]*schema.Schema{
        "access_key": {
            Type:        schema.TypeString,
            Required:    true,
            DefaultFunc: schema.EnvDefaultFunc("NCLOUD_ACCESS_KEY", nil),
        },
        "secret_key": {
            Type:        schema.TypeString,
            Required:    true,
            DefaultFunc: schema.EnvDefaultFunc("NCLOUD_SECRET_KEY", nil),
        },
        "region": {
            Type:        schema.TypeString,
            Required:    true,
            DefaultFunc: schema.EnvDefaultFunc("NCLOUD_REGION", nil),
        },
        /.../
}
```

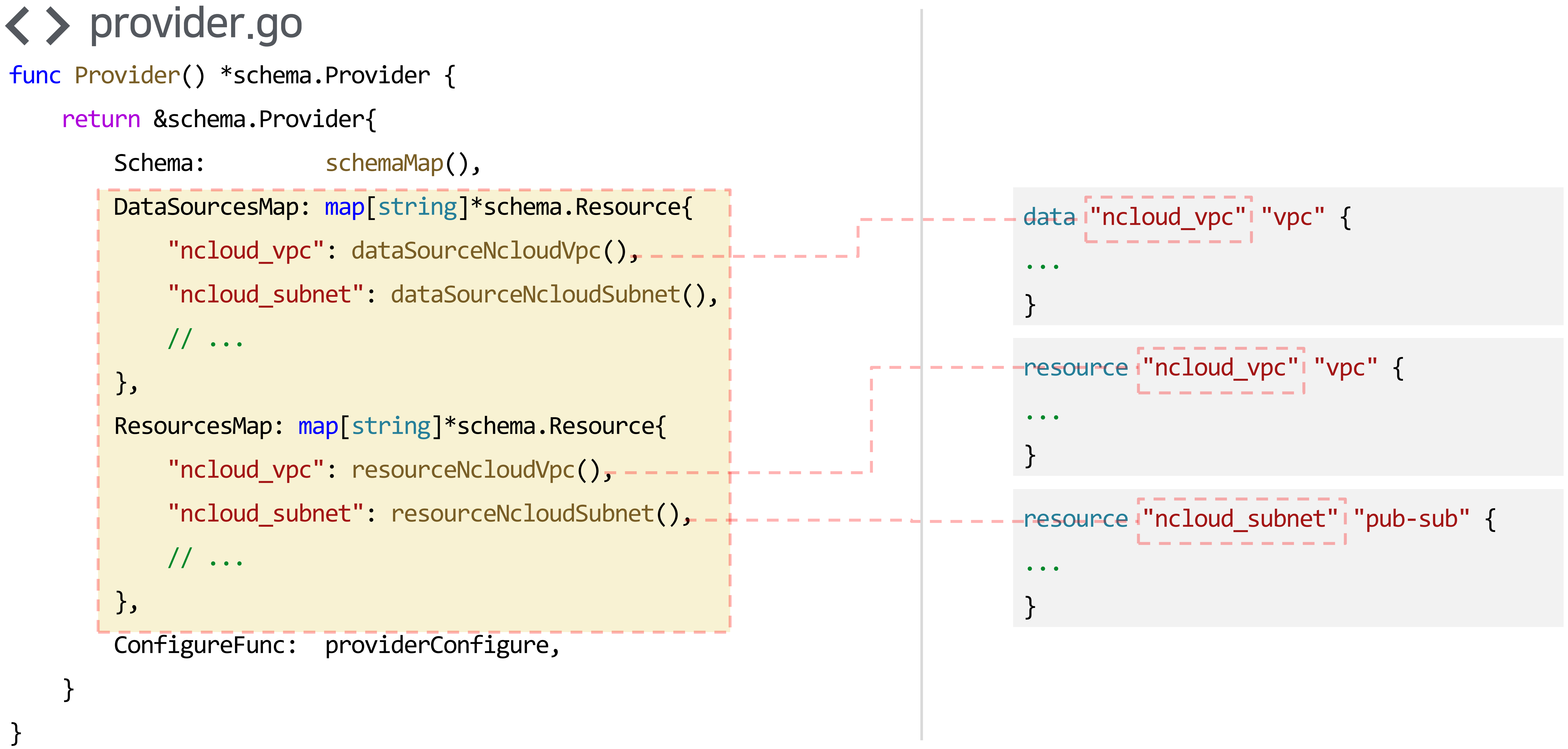
Terraform config (*.tf)

```hcl
provider "ncloud" {
  access_key  = "ACCESS_KEY"
  secret_key  = "SECRET_KEY"
  region      = "KR"
  support_vpc = true
}

resource "ncloud_vpc" "vpc" {
  name            = "deview-vpc"
  ipv4_cidr_block = "10.0.0.0/16"
}
```

Provider를 정의하자

Resource와 Data Source들을 정의

제공할 리소스들을 map으로 정의



provider.go

```go
func Provider() *schema.Provider {
    return &schema.Provider{
        Schema:         schemaMap(),
        DataSourcesMap: map[string]*schema.Resource{
            "ncloud_vpc": dataSourceNcloudVpc(),
            "ncloud_subnet": dataSourceNcloudSubnet(),
            // ...
        },
        ResourcesMap: map[string]*schema.Resource{
            "ncloud_vpc": resourceNcloudVpc(),
            "ncloud_subnet": resourceNcloudSubnet(),
            // ...
        },
        ConfigureFunc:  providerConfigure,
    }
}
```

```
data "ncloud_vpc" "vpc" {
...
}
```

```
resource "ncloud_vpc" "vpc" {
...
}
```

```
resource "ncloud_subnet" "pub-sub" {
...
}
```

```
/docs
/examples
/ncloud
∟ provider.go
∟ provider_test.go
∟ data_source_ncloud_vpc.go
∟ data_source_ncloud_vpc_test.go
∟ resource_ncloud_vpc.go
∟ resource_ncloud_vpc_test.go
∟ resource_ncloud_subnet.go
∟ resource_ncloud_subnet _test.go
...
```

Provider를 정의하자

ConfigureFunc

- API 호출에 사용되는 포함된 라이브러리의 **인증** 및 **초기화**
- **메타정보** 설정 (리젼 코드)

provider.go

```go
func Provider() *schema.Provider {
    return &schema.Provider{
        Schema:         schemaMap(),
        DataSourcesMap: map[string]*schema.Resource{
            "ncloud_vpc": dataSourceNcloudVpc(),
            "ncloud_subnet": dataSourceNcloudSubnet(),
            // ...
        },
        ResourcesMap: map[string]*schema.Resource{
            "ncloud_vpc": resourceNcloudVpc(),
            "ncloud_subnet": resourceNcloudSubnet(),
            // ...
        },
        ConfigureFunc:  providerConfigure,
    }
}
```

```go
func providerConfigure(d *schema.ResourceData) (interface{}, error) {
    providerConfig := ProviderConfig{
        RegionCode: d.Get("region").(string),
    }

    config := Config{
        AccessKey: d.Get("access_key").(string),
        SecretKey: d.Get("secret_key").(string),
    }

    if client, err := config.Client(); err != nil {
        return nil, err
    } else {
        providerConfig.Client = client
    }
    ...

    return &providerConfig, nil
}
```

Terraform config (*.tf)

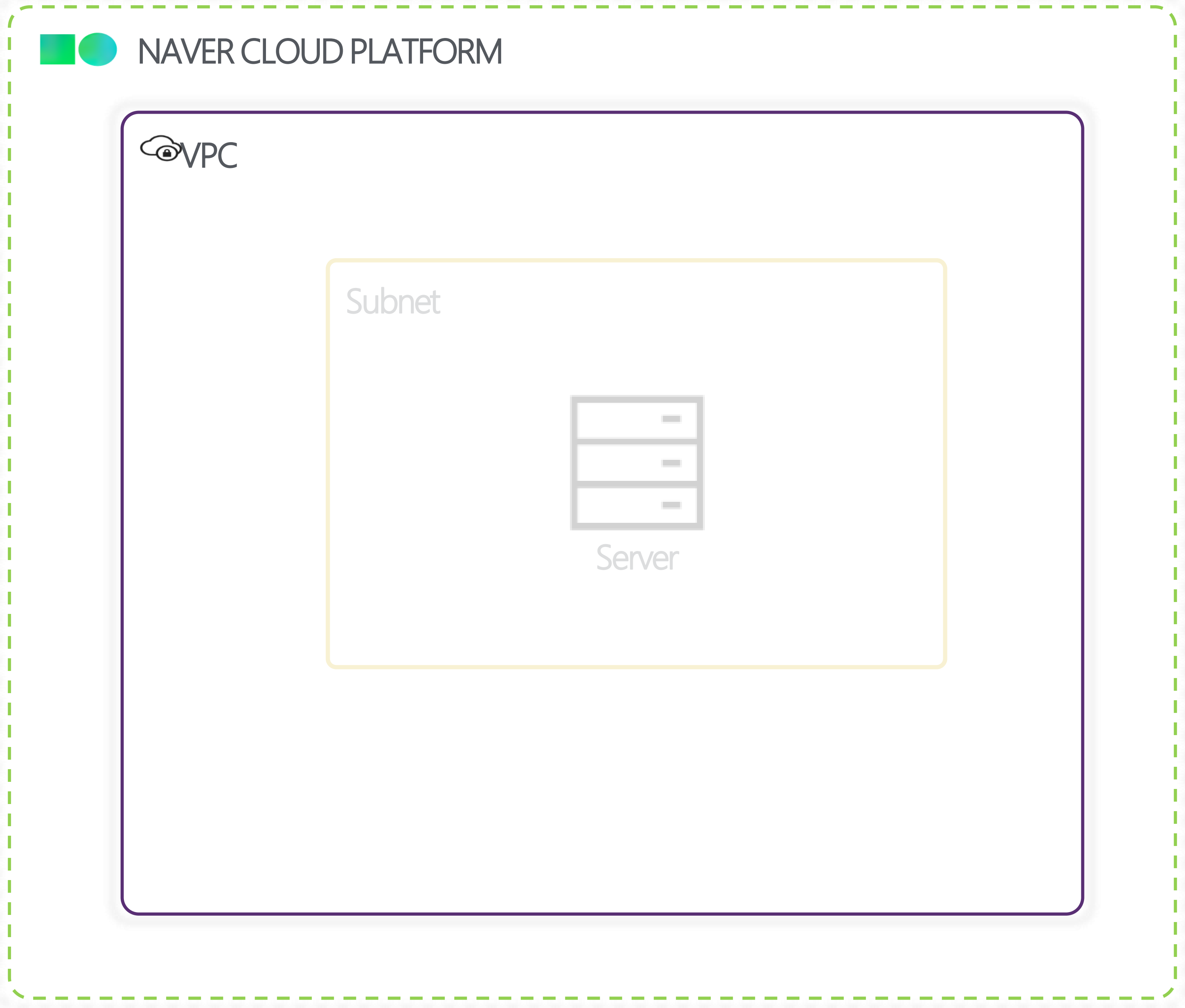
```hcl
provider "ncloud" {
  access_key  = "ACCESS_KEY"
  secret_key  = "SECRET_KEY"
  region      = "KR"
  support_vpc = true
}

resource "ncloud_vpc" "vpc" {
  name            = "deview-vpc"
  ipv4_cidr_block = "10.0.0.0/16"
}
```

Resource 개발하기 (VPC)

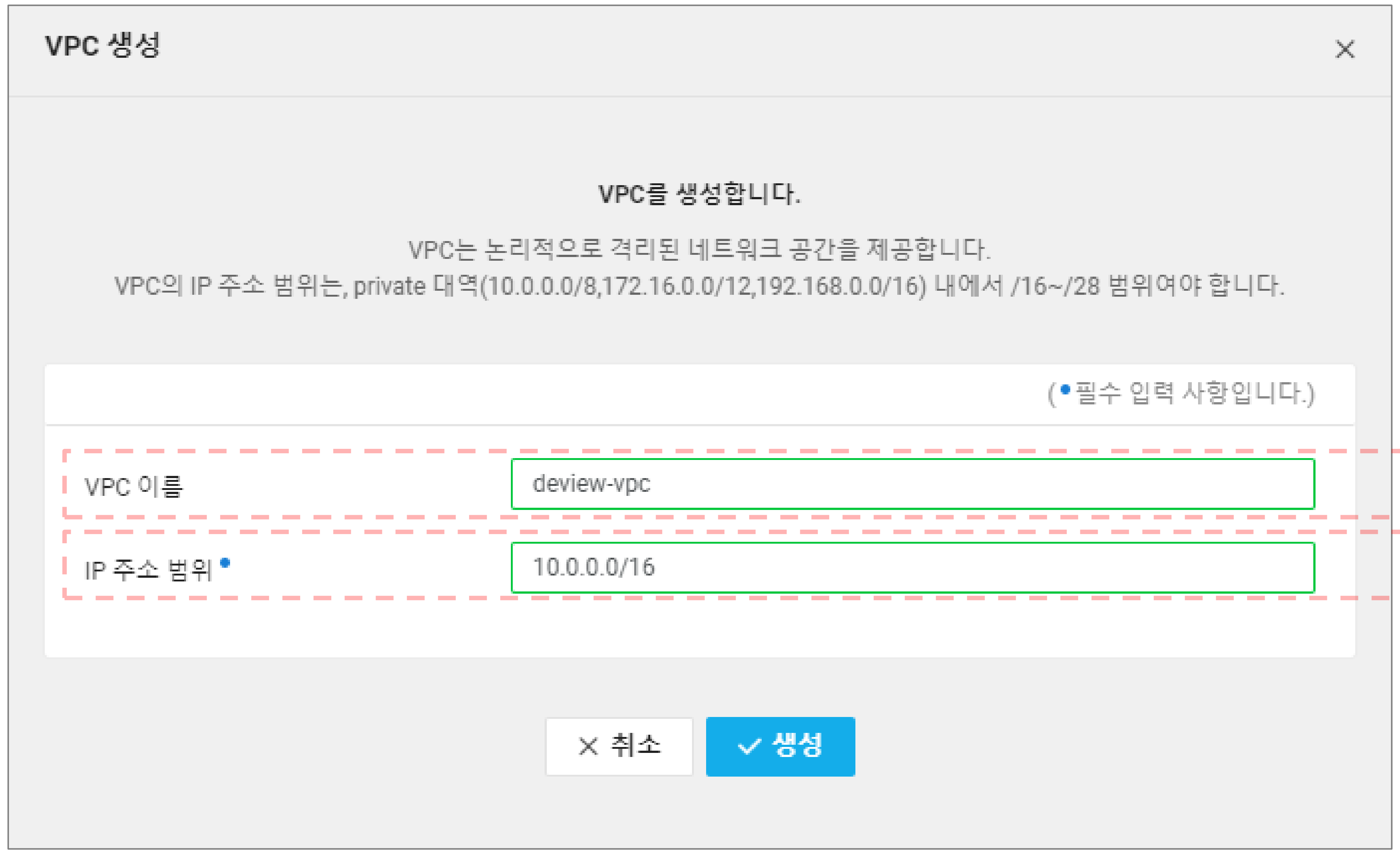
Schema Attributes과 Type을 정의

VPC는 이름과 IP주소 범위를 String형 입력으로 받음

Provider



Console UI



VPC 생성

VPC를 생성합니다.

VPC는 논리적으로 격리된 네트워크 공간을 제공합니다.
VPC의 IP 주소 범위는, private 대역(10.0.0.0/8,172.16.0.0/12,192.168.0.0/16) 내에서 /16~/28 범위여야 합니다.

(•필수 입력 사항입니다.)

VPC 이름	deview-vpc
IP 주소 범위•	10.0.0.0/16

Terraform config (*.tf)

```
provider "ncloud" {
  access_key  = "ACCESS_KEY"
  secret_key  = "SECRET_KEY"
  region      = "KR"
  support_vpc = true
}

resource "ncloud_vpc" "vpc" {
  name            = "deview-vpc"
  ipv4_cidr_block = "10.0.0.0/16"
}
```

String Type

Resource 개발하기 (VPC)

Schema Attributes과 Type을 정의

- Attributes는 name과 ipv4_cidr_block로 지정
- Types은 모두 TypeString으로

Schema Types
- TypeString
- TypeInt
- TypeFloat
- TypeMap
- TypeList
- TypeSet

resource_ncloud_vpc.go

```go
func resourceNcloudVpc() *schema.Resource {
    return &schema.Resource{
        CreateContext: resourceNcloudVpcCreate,
        ReadContext:   resourceNcloudVpcRead,
        UpdateContext: resourceNcloudVpcUpdate,
        DeleteContext: resourceNcloudVpcDelete,
        Importer: &schema.ResourceImporter{
            State: schema.ImportStatePassthrough,
        },
        Schema: map[string]*schema.Schema{…},
    }
}
```

```go
"name": {
    Type:             schema.TypeString,
    Optional:         true,
    Computed:         true,
    ForceNew:         true,
    ValidateDiagFunc: ToDiagFunc(validateInstanceName),
},
"ipv4_cidr_block": {
    Type:             schema.TypeString,
    Required:         true,
    ForceNew:         true,
    ValidateDiagFunc: ToDiagFunc(validation.IsCIDRNetwork(16, 28)),
},
"vpc_no": {
    Type:     schema.TypeString,
    Computed: true,
},
```

Terraform config (*.tf)

```hcl
provider "ncloud" {
  access_key  = "ACCESS_KEY"
  secret_key  = "SECRET_KEY"
  region      = "KR"
  support_vpc = true
}

resource "ncloud_vpc" "vpc" {
  name            = "deview-vpc"
  ipv4_cidr_block = "10.0.0.0/16"
}
```

Resource 개발하기 (VPC)

Schema Behaviors정의

- name은 필수 값이 아니기 때문에 Optional
- ipv4_cidr_block는 필수라서 Required
- ipv4_cidr_block는 수정을 지원하지 않기 때문에 ForceNew (재생성)

Primitive Behaviors
- Optional
- Required
- Default
- Computed
- ForceNew

resource_ncloud_vpc.go

```go
func resourceNcloudVpc() *schema.Resource {
    return &schema.Resource{
        CreateContext: resourceNcloudVpcCreate,
        ReadContext:   resourceNcloudVpcRead,
        UpdateContext: resourceNcloudVpcUpdate,
        DeleteContext: resourceNcloudVpcDelete,
        Importer: &schema.ResourceImporter{
            State: schema.ImportStatePassthrough,
        },
        Schema: map[string]*schema.Schema{…},
    }
}
```

```go
"name": {
    Type:             schema.TypeString,
    Optional:         true,
    Computed:         true,
    ForceNew:         true,
    ValidateDiagFunc: ToDiagFunc(validateInstanceName),
},
"ipv4_cidr_block": {
    Type:             schema.TypeString,
    Required:         true,
    ForceNew:         true,
    ValidateDiagFunc: ToDiagFunc(validation.IsCIDRNetwork(16, 28)),
},
"vpc_no": {
    Type:     schema.TypeString,
    Computed: true,
},
```

Terraform config (*.tf)

```hcl
provider "ncloud" {
  access_key  = "ACCESS_KEY"
  secret_key  = "SECRET_KEY"
  region      = "KR"
  support_vpc = true
}

resource "ncloud_vpc" "vpc" {
  name            = "deview-vpc"
  ipv4_cidr_block = "10.0.0.0/16"
}
```

Resource 개발하기 (VPC)

Schema Behaviors정의

- vpc_no는 생성된 후 알 수 있기 때문에 Computed
- name의 입력하지 않을 경우
  자동으로 이름을 생성하기 때문에 Computed

Primitive Behaviors
- Optional
- Required
- Default
- Computed
- ForceNew

resource_ncloud_vpc.go

```go
func resourceNcloudVpc() *schema.Resource {
    return &schema.Resource{
        CreateContext: resourceNcloudVpcCreate,
        ReadContext:   resourceNcloudVpcRead,
        UpdateContext: resourceNcloudVpcUpdate,
        DeleteContext: resourceNcloudVpcDelete,
        Importer: &schema.ResourceImporter{
            State: schema.ImportStatePassthrough,
        },
        Schema: map[string]*schema.Schema{…},
    }
}
```

```go
"name": {
    Type:             schema.TypeString,
    Optional:         true,
    Computed:         true,
    ForceNew:         true,
    ValidateDiagFunc: ToDiagFunc(validateInstar
},
"ipv4_cidr_block": {
    Type:             schema.TypeString,
    Required:         true,
    ForceNew:         true,
    ValidateDiagFunc: ToDiagFunc(validation.Is(
},
"vpc_no": {
    Type:     schema.TypeString,
    Computed: true,
},
```

Terraform config (*.tf)

```hcl
resource "ncloud_vpc" "vpc" {
  ipv4_cidr_block = "10.0.0.0/16"
}

resource "ncloud_subnet" "pub-sub" {
  vpc_no         = ncloud_vpc.vpc.vpc_no
  subnet         = "10.0.1.0/24"
  zone           = "KR-2"
  network_acl_no = ncloud_vpc.vpc.default_network_acl_no
  subnet_type    = "PUBLIC"
}
```

Resource 개발하기 (VPC)

Schema Behaviors정의

name의 경우 인스턴스 명 규칙에 맞는지, ValidateFunc사용

Function Behaviors
- DiffSuppressFunc
- DefaultFunc
- StateFunc
- ValidateFunc

resource_ncloud_vpc.go

```go
func resourceNcloudVpc() *schema.Resource {
    return &schema.Resource{
        CreateContext: resourceNcloudVpcCreate,
        ReadContext:   resourceNcloudVpcRead,
        UpdateContext: resourceNcloudVpcUpdate,
        DeleteContext: resourceNcloudVpcDelete,
        Importer: &schema.ResourceImporter{
            State: schema.ImportStatePassthrough,
        },
        Schema: map[string]*schema.Schema{…},
    }
}
```

```go
"name": {
    Type:             schema.TypeString,
    Optional:         true,
    Computed:         true,
    ForceNew:         true,
    ValidateDiagFunc: ToDiagFunc(validateInstanceName),
},
"ipv4_cidr_block": {
    Type:             schema.TypeString,
    Required:         true,
    ForceNew:         true,
    ValidateDiagFunc: ToDiagFunc(validation.IsCIDRNetwork(16, 28)),
},
"vpc_no": {
    Type:     schema.TypeString,
    Computed: true,
},
```

Terraform config (*.tf)

```hcl
provider "ncloud" {
  access_key  = "ACCESS_KEY"
  secret_key  = "SECRET_KEY"
  region      = "KR"
  support_vpc = true
}

resource "ncloud_vpc" "vpc" {
  name            = "deview-vpc"
  ipv4_cidr_block = "10.0.0.0/16"
}
```

Resource 개발하기 (VPC)

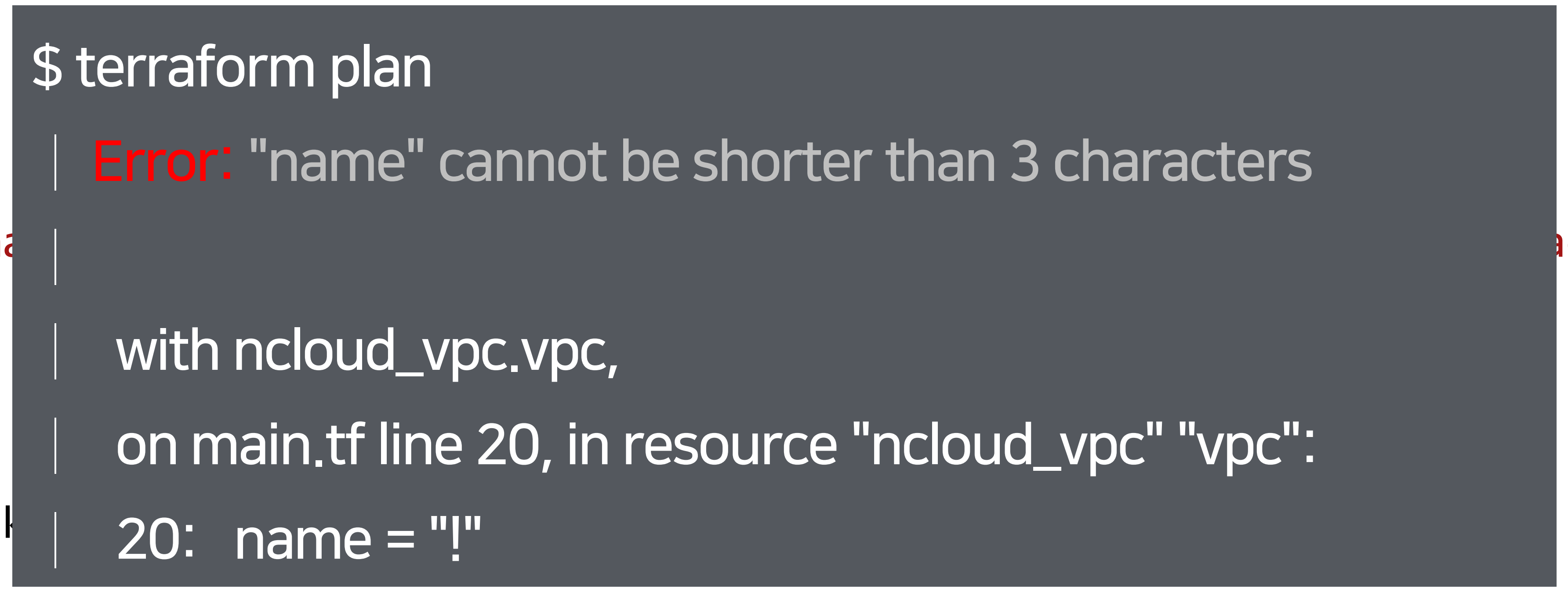
N DEVIEW 2021

VadliateFunc

사용자 입력을 검증 할 수 있으며, plan단계에서 검증 가능

(IPv4 형식, Min, Max, 인스턴스 명 등)

```
func validateInstanceName(v interface{}, k string) (ws []string, errors []error) {
    value := v.(string)

    if len(value) < 3 {
        errors = append(errors, fmt.Errorf(
            "%q cannot be shorter than 3 characters", k))
    }

    if len(value) > 30 {
        errors = append(errors, fmt.Errorf(
            "%q cannot be longer than 30 characters", k))
    }

    if !regexp.MustCompile(`^[a-z][a-z0-9-]*$`).MatchString(value)
        errors = append(errors, fmt.Errorf(
            "%s can only lowercase letters, numbers and special cha                                  acter", k))
    }

    if regexp.MustCompile(`.*(-|_)$`).MatchString(value) {
        errors = append(errors, fmt.Errorf(
            "%q must end with an alphabetic character or number", k
    }

    return
}
```



```
$ terraform plan
│ Error: "name" cannot be shorter than 3 characters
│
│   with ncloud_vpc.vpc,
│   on main.tf line 20, in resource "ncloud_vpc" "vpc":
│   20:   name = "!"
```

Terraform config (*.tf)

```
provider "ncloud" {
  access_key  = "ACCESS_KEY"
  secret_key  = "SECRET_KEY"
  region      = "KR"
  support_vpc = true
}

resource "ncloud_vpc" "vpc" {
  name            = "!"
  ipv4_cidr_block = "10.0.0.0/16"
}
```

Resource 개발하기 (VPC)

CRUD Func 구현

- Terraform core가 해당 구현체를 통해 인프라를 **생성/수정/삭제** 함
- Read operation을 통해 **state**(.tfstate)를 **refresh** 함

< > resource_ncloud_vpc.go

```go
func resourceNcloudVpc() *schema.Resource {
    return &schema.Resource{
        CreateContext: resourceNcloudVpcCreate,
        ReadContext:   resourceNcloudVpcRead,
        UpdateContext: resourceNcloudVpcUpdate,
        DeleteContext: resourceNcloudVpcDelete,
        Importer: &schema.ResourceImporter{
            State: schema.ImportStatePassthrough,
        },
        Schema: map[string]*schema.Schema{…},
    }
}
```

```go
func resourceNcloudVpcCreate(ctx context.Context, d *schema.ResourceData, meta interface{}) diag.Diagnostics {
  // /vpc/v2/createVpc 를 호출
}

func resourceNcloudVpcRead(ctx context.Context, d *schema.ResourceData, meta interface{}) diag.Diagnostics {
 // /vpc/v2/getVpcDetail 를 호출
}

func resourceNcloudVpcUpdate(ctx context.Context, d *schema.ResourceData, meta interface{}) diag.Diagnostics {
 // /vpc/v2/updateVpc 를 호출
}

func resourceNcloudVpcDelete(ctx context.Context, d *schema.ResourceData, meta interface{}) diag.Diagnostics {
 // /vpc/v2/deleteVpc 를 호출
}
```

Resource 개발하기 (VPC)

CRUD Func Parameters

ConfigureFunc(Provider)에서 정의 한 meta정보를 사용 할 수 있다. (리젼 코드)

resource_ncloud_vpc.go

```go
func resourceNcloudVpc() *schema.Resource {
    return &schema.Resource{
        CreateContext: resourceNcloudVpcCreate,
        ReadContext:   resourceNcloudVpcRead,
        UpdateContext: resourceNcloudVpcUpdate,
        DeleteContext: resourceNcloudVpcDelete,
        Importer: &schema.ResourceImporter{
            State: schema.ImportStatePassthrough,
        },
        Schema: map[string]*schema.Schema{…},
    }
}
```

```go
func resourceNcloudVpcCreate(ctx context.Context, d *schema.ResourceData, meta interface{}) diag.Diagnostics {
    config := meta.(*ProviderConfig)

    reqParams := &vpc.CreateVpcRequest{
        RegionCode:    &config.RegionCode,
        VpcName:       ncloud.String(d.Get("name").(string)),
        Ipv4CidrBlock: ncloud.String(d.Get("ipv4_cidr_block").(string)),
    }

    resp, err := config.Client.vpc.V2Api.CreateVpc(ctx, reqParams)
    if err != nil {
        return diag.FromErr(err)
    }

    vpcInstance := resp.VpcList[0]
    d.SetId(*vpcInstance.VpcNo)

    return resourceNcloudVpcRead(ctx, d, meta)
}
```

Terraform config (*.tf)

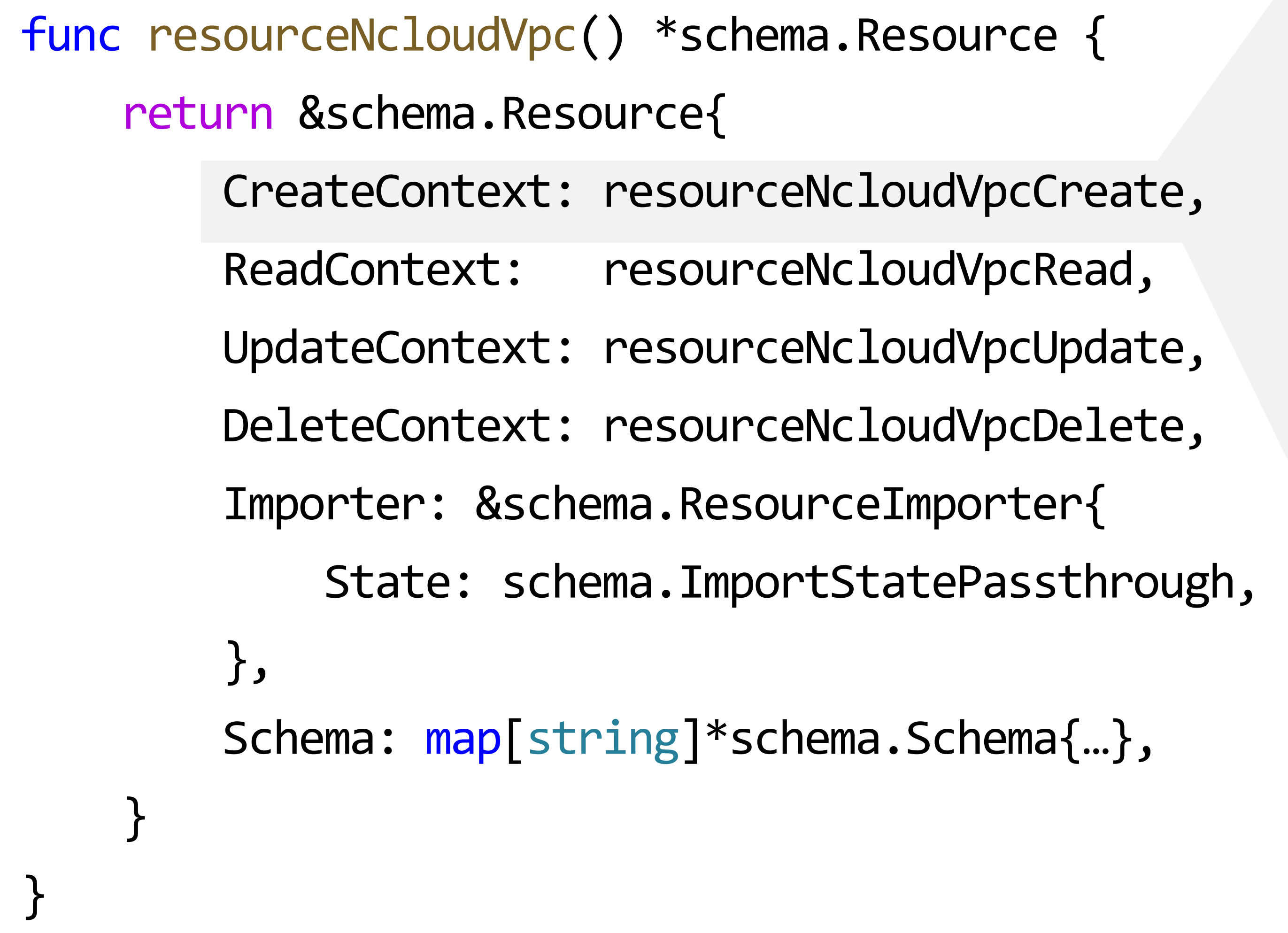
```hcl
provider "ncloud" {
  access_key  = "ACCESS_KEY"
  secret_key  = "SECRET_KEY"
  region      = "KR"
  support_vpc = true
}

resource "ncloud_vpc" "vpc" {
  name            = "deview-vpc"
  ipv4_cidr_block = "10.0.0.0/16"
}
```

Resource 개발하기 (VPC)

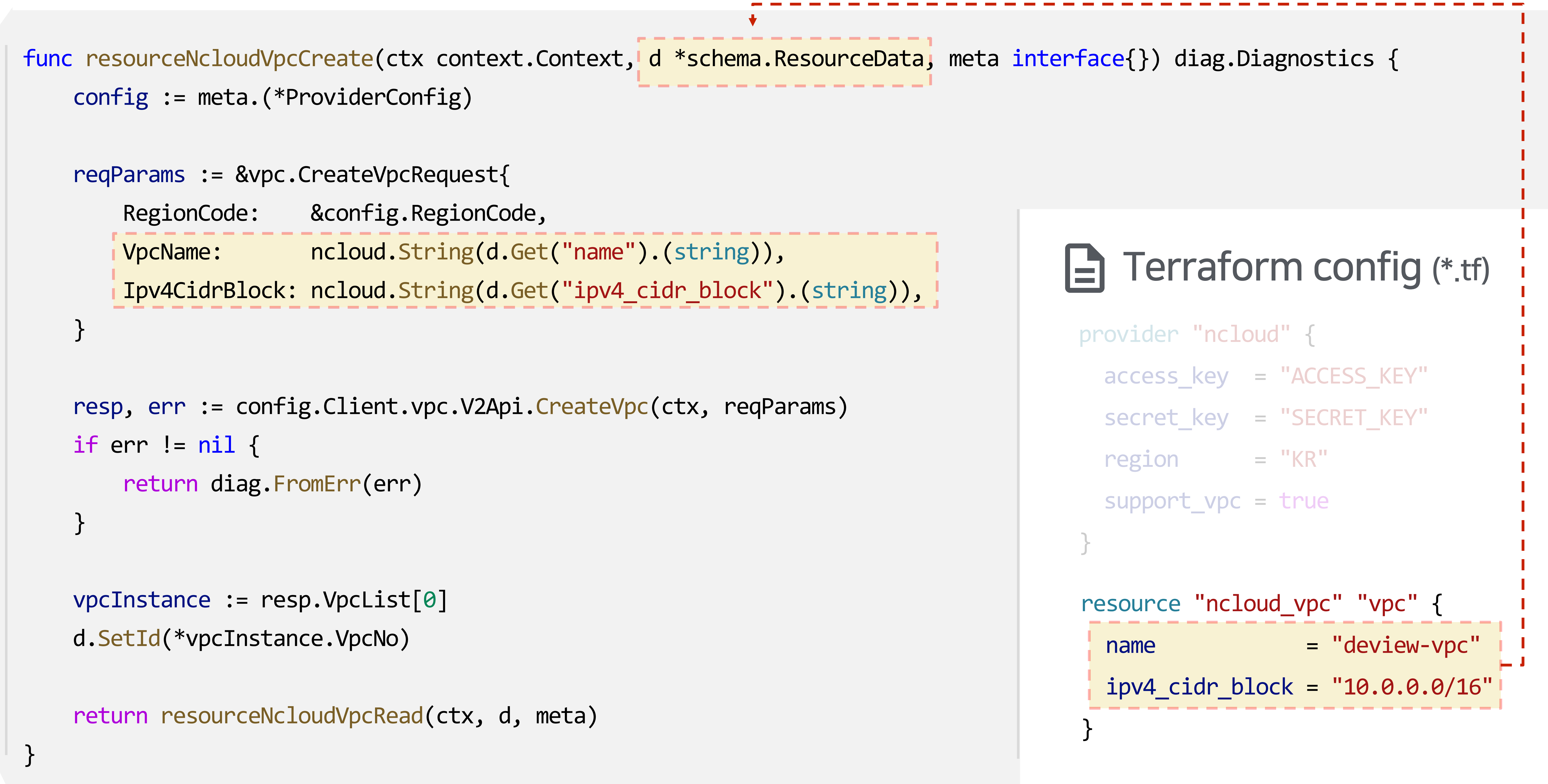
CRUD Func Parameters

d.Get("name")를 통해 Terraform code의 입력을 받음

resource_ncloud_vpc.go



```go
func resourceNcloudVpc() *schema.Resource {
    return &schema.Resource{
        CreateContext: resourceNcloudVpcCreate,
        ReadContext:   resourceNcloudVpcRead,
        UpdateContext: resourceNcloudVpcUpdate,
        DeleteContext: resourceNcloudVpcDelete,
        Importer: &schema.ResourceImporter{
            State: schema.ImportStatePassthrough,
        },
        Schema: map[string]*schema.Schema{…},
    }
}
```



```go
func resourceNcloudVpcCreate(ctx context.Context, d *schema.ResourceData, meta interface{}) diag.Diagnostics {
    config := meta.(*ProviderConfig)

    reqParams := &vpc.CreateVpcRequest{
        RegionCode:    &config.RegionCode,
        VpcName:       ncloud.String(d.Get("name").(string)),
        Ipv4CidrBlock: ncloud.String(d.Get("ipv4_cidr_block").(string)),
    }

    resp, err := config.Client.vpc.V2Api.CreateVpc(ctx, reqParams)
    if err != nil {
        return diag.FromErr(err)
    }

    vpcInstance := resp.VpcList[0]
    d.SetId(*vpcInstance.VpcNo)

    return resourceNcloudVpcRead(ctx, d, meta)
}
```

Terraform config (*.tf)

```hcl
provider "ncloud" {
  access_key  = "ACCESS_KEY"
  secret_key  = "SECRET_KEY"
  region      = "KR"
  support_vpc = true
}

resource "ncloud_vpc" "vpc" {
  name            = "deview-vpc"
  ipv4_cidr_block = "10.0.0.0/16"
}
```

Resource 개발하기 (VPC)

Create 구현

- 인프라를 생성하고, ID를 설정
- 마지막으로 Read함수 를 호출해서 state(.tfstate)를 업데이트 할 수 있도록 한다

resource_ncloud_vpc.go

```go
func resourceNcloudVpc() *schema.Resource {
    return &schema.Resource{
        CreateContext: resourceNcloudVpcCreate,
        ReadContext:   resourceNcloudVpcRead,
        UpdateContext: resourceNcloudVpcUpdate,
        DeleteContext: resourceNcloudVpcDelete,
        Importer: &schema.ResourceImporter{
            State: schema.ImportStatePassthrough,
        },
        Schema: map[string]*schema.Schema{…},
    }
}
```

```go
func resourceNcloudVpcCreate(ctx context.Context, d *schema.ResourceData, meta interface{}) diag.Diagnostics {
    config := meta.(*ProviderConfig)

    reqParams := &vpc.CreateVpcRequest{
        RegionCode:    &config.RegionCode,
        VpcName:       ncloud.String(d.Get("name").(string)),
        Ipv4CidrBlock: ncloud.String(d.Get("ipv4_cidr_block").(string)),
    }

    resp, err := config.Client.vpc.V2Api.CreateVpc(ctx, reqParams)
    if err != nil {
        return diag.FromErr(err)
    }

    vpcInstance := resp.VpcList[0]
    d.SetId(*vpcInstance.VpcNo)

    return resourceNcloudVpcRead(ctx, d, meta)
}
```

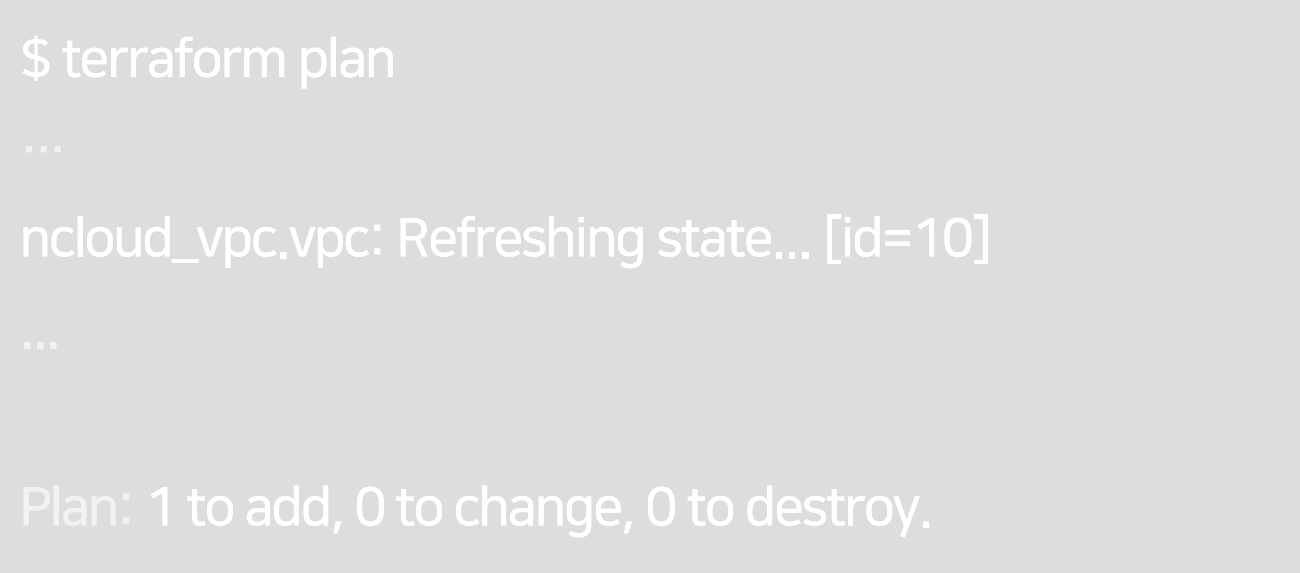
Terraform config (*.tf)

```hcl
provider "ncloud" {
  access_key  = "ACCESS_KEY"
  secret_key  = "SECRET_KEY"
  region      = "KR"
  support_vpc = true
}

resource "ncloud_vpc" "vpc" {
  name            = "deview-vpc"
  ipv4_cidr_block = "10.0.0.0/16"
}
```

Resource 개발하기 (VPC)

Read 구현

- state(.tfstate)를 sync 하는데 사용
- ID는 Unique한 값이어야 함



```
$ terraform plan
…
ncloud_vpc.vpc: Refreshing state... [id=10]
…

Plan: 1 to add, 0 to change, 0 to destroy.
```

resource_ncloud_vpc.go

```go
func resourceNcloudVpc() *schema.Resource {
    return &schema.Resource{
        CreateContext: resourceNcloudVpcCreate,
        ReadContext:   resourceNcloudVpcRead,
        UpdateContext: resourceNcloudVpcUpdate,
        DeleteContext: resourceNcloudVpcDelete,
        Importer: &schema.ResourceImporter{
            State: schema.ImportStatePassthrough,
        },
        Schema: map[string]*schema.Schema{…},
    }
}
```

```go
func resourceNcloudVpcRead(ctx context.Context, d *schema.ResourceData, meta interface{}) diag.Diagnostics {
    config := meta.(*ProviderConfig)

    instance, err := getVpcInstance(config, d.Id())
    if err != nil {
        return diag.FromErr(err)
    }

    if instance == nil {
        d.SetId("")
        return nil
    }

    d.SetId(*instance.VpcNo)
    d.Set("vpc_no", instance.VpcNo)
    d.Set("name", instance.VpcName)
    d.Set("ipv4_cidr_block", instance.Ipv4CidrBlock)

    return nil
}
```

Resource 개발하기 (VPC)

Update 구현

- d.HasChange("name") 를 통해 변경여부를 알 수 있다
- 마찬가지로 Read함수 를 호출해서 state(.tfstate)를 **업데이트** 할 수 있도록 한다

resource_ncloud_vpc.go

```go
func resourceNcloudVpc() *schema.Resource {
    return &schema.Resource{
        CreateContext: resourceNcloudVpcCreate,
        ReadContext:   resourceNcloudVpcRead,
        UpdateContext: resourceNcloudVpcUpdate,
        DeleteContext: resourceNcloudVpcDelete,
        Importer: &schema.ResourceImporter{
            State: schema.ImportStatePassthrough,
        },
        Schema: map[string]*schema.Schema{…},
    }
}
```

```go
func resourceNcloudVpcUpdate(ctx context.Context, d *schema.ResourceData, meta
interface{}) diag.Diagnostics {
    config := meta.(*ProviderConfig)

    if d.HasChange("name") {
        reqParams := &vpc.UpdateVpcRequest{
            RegionCode: &config.RegionCode,
            VpcNo:      ncloud.String(d.Id().(string)),
            Name:       ncloud.String(d.Get("name").(string))
        }

        resp, err := config.Client.vpc.V2Api.UpdateVpc(ctx, reqParams)
        if err != nil {
            return diag.FromErr(err)
        }
    }
    return resourceNcloudVpcRead(ctx, d, meta)
}
```

Terraform config (*.tf)

```hcl
resource "ncloud_vpc" "vpc" {
  name            = "vpc"
  ipv4_cidr_block = "10.0.0.0/16"
}
```

```hcl
resource "ncloud_vpc" "vpc" {
  name            = "deview"
  ipv4_cidr_block = "10.0.0.0/16"
}
```

Resource 개발하기 (VPC)

DEVIEW 2021

Delete 구현

- Terraform **destroy**또는 Diff결과 **삭제** 될 경우 호출
- 속성이 **ForceNew**인 경우, **삭제 후 생성** 하기 위해 사용

resource_ncloud_vpc.go

```go
func resourceNcloudVpc() *schema.Resource {
    return &schema.Resource{
        CreateContext: resourceNcloudVpcCreate,
        ReadContext:   resourceNcloudVpcRead,
        UpdateContext: resourceNcloudVpcUpdate,
        DeleteContext: resourceNcloudVpcDelete,
        Importer: &schema.ResourceImporter{
            State: schema.ImportStatePassthrough,
        },
        Schema: map[string]*schema.Schema{…},
    }
}
```

```go
func resourceNcloudVpcDelete(ctx context.Context, d *schema.ResourceData, meta
interface{}) diag.Diagnostics {
    config := meta.(*ProviderConfig)

    reqParams := &vpc.DeleteVpcRequest{
        RegionCode: &config.RegionCode,
        VpcNo:      ncloud.String(d.Id().(string)),
    }

    resp, err := config.Client.vpc.V2Api.DeleteVpc(ctx, reqParams)
    if err != nil {
        return diag.FromErr(err)
    }

    return nil
}
```

Terraform config (*.tf)

```hcl
resource "ncloud_vpc" "vpc" {
  name            = "vpc"
  ipv4_cidr_block = "10.0.0.0/16"
}
```

```hcl
# resource "ncloud_vpc" "vpc" {
#   name            = "deview"
#   ipv4_cidr_block = "10.0.0.0/16"
# }
```

Terraform provider 만들기

Testing

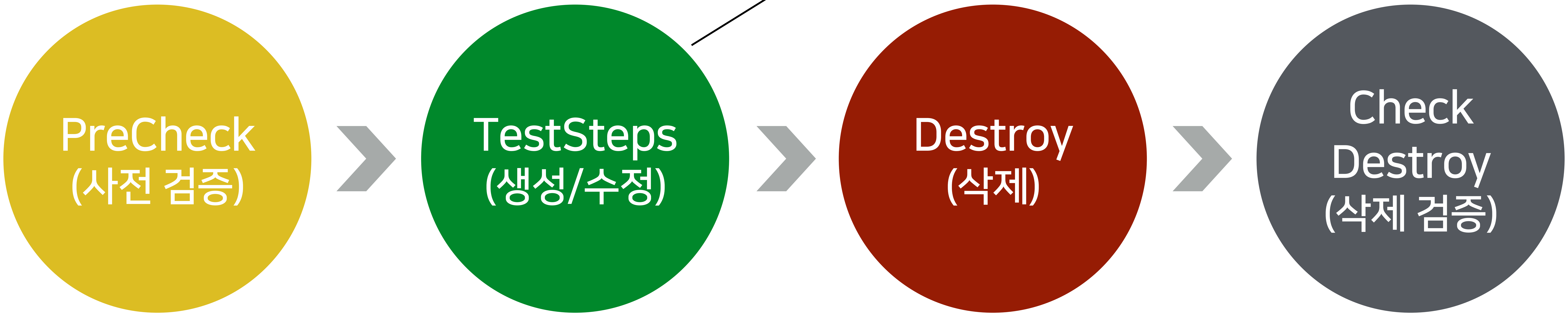
- PreCheck: provider 리젼 설정 여부, 입력 Validation
- TestSteps: 실제 인프라 생성 또는 변경 테스트
- Destory: 테스트가 종료된 후, 제거하는 역할
- CheckDestory: 잘 삭제가 되었는지 검증

```
resource "ncloud_vpc" "vpc" {
  name            = "vpc"
  ipv4_cidr_block = "10.0.0.0/16"
}
```

Step1

```
resource "ncloud_vpc" "vpc" {
  name            = "deview"
  ipv4_cidr_block = "10.0.0.0/16"
}
```

Step2



Terraform provider 만들기

Testing

```
/docs
/examples
/ncloud
∟ provider.go
∟ provider_test.go
∟ data_source_ncloud_vpc.go
∟ data_source_ncloud_vpc_test.go
∟ resource_ncloud_vpc.go
∟ resource_ncloud_vpc_test.go
∟ resource_ncloud_subnet.go
∟ resource_ncloud_subnet _test.go
...
```

resource_ncloud_vpc_test.go

```
func TestAccResourceNcloudVpc_basic(t *testing.T) {
    var vpc vpc.Vpc
    rInt := rand.Intn(16)
    cidr := fmt.Sprintf("10.%d.0.0/16", rInt)
    name := fmt.Sprintf("test-vpc-basic-%s", acctest.RandString(5))
    resourceName := "ncloud_vpc.test"

    resource.Test(t, resource.TestCase{
        PreCheck:     func() { testAccPreCheck(t) },
        Providers:    testAccProviders,
        CheckDestroy: testAccCheckVpcDestroy,
        Steps: []resource.TestStep{
            {
                Config: testAccDataSourceNcloudVpcConfig(name, cidr),
                Check: resource.ComposeTestCheckFunc(
                    testAccCheckVpcExists(resourceName, &vpc),
                    resource.TestCheckResourceAttr(resourceName, "ipv4_cidr_block", cidr),
                    resource.TestCheckResourceAttr(resourceName, "name", name),
                    resource.TestMatchResourceAttr(resourceName, "vpc_no", regexp.MustCompile(`^\d+$`)),
                ),
            },
        },
    })
```

```
func testAccDataSourceNcloudVpcConfig(name, cidr string) string {
    return fmt.Sprintf(`
resource "ncloud_vpc" "test" {
  name              = "%s"
  ipv4_cidr_block   = "%s"
}

data "ncloud_vpc" "by_id" {
  id = ncloud_vpc.test.id
}

data "ncloud_vpc" "by_filter" {
  filter {
    name = "vpc_no"
    values = [ncloud_vpc.test.id]
  }
}
`, name, cidr)
}
```

Terraform provider 만들기

Testing

- resource.ParelleTest 와 같은 **병렬 테스트**도 지원 (but. 동시성 고려)
- TF_ACC=true 환경 변수를 통해,
  실제 인프라에 반영되는 Acceptance Tests수행
- dev_overrides 를 사용하면, required_providers 설정하고 사용 시
  Binary파일 SUM체크 무시

Terraform provider 만들기

Debugging

- TF_LOG=DEBUG 를 통해 로깅 하자
- 디버깅을 위해 각 operation 마다 Logging 하는 것 이 중요
- 에러 발생 시 request와 response가 잘 나오도록

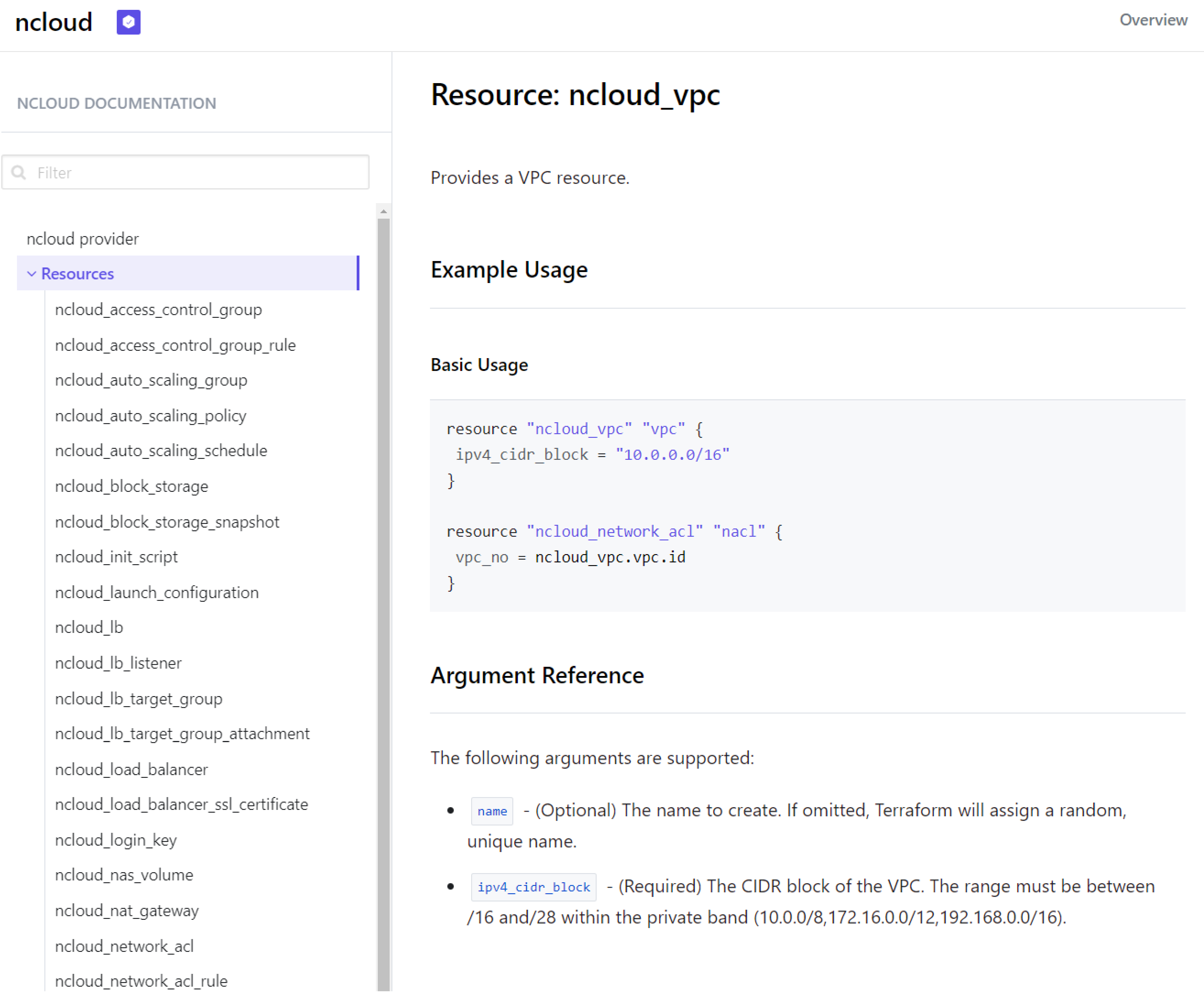
```
2021/02/24 12:13:25 [DEBUG] Waiting for state to become: [RUN]
2021/02/24 12:13:28 [INFO] GetVpcDetail response={"requestId":"f5b465cd-4218-4542-954c-b4dbb617f787","returnCode":"0","returnMessage":"success","totalRows":1,"vpcList":[{"vpcNo":"5068","vpc
2021/02/24 12:13:48 [INFO] GetNetworkAclList response={"requestId":"f0b05435-7819-4741-a3b1-0bbdb4047958","returnCode":"0","returnMessage":"success","totalRows":1,"networkAclList":[{"networ
ACL","createDate":"2021-02-24T12:13:25+0900","isDefault":true}]}
2021/02/24 12:13:48 [INFO] getDefaultAccessControlGroup params={"regionCode":"KR","vpcNo":"5068"}
2021/02/24 12:13:48 [INFO] getDefaultAccessControlGroup response={"requestId":"6dc80293-9cf4-4e07-96ad-5a8f547ad976","returnCode":"0","returnMessage":"success","totalRows":1,"accessControlG
acg","isDefault":true,"vpcNo":"5068","accessControlGroupStatus":{"code":"RUN","codeName":"run"},"accessControlGroupDescription":"VPC [v177d206ef58] default ACG"}]}
2021/02/24 12:13:48 [INFO] getDefaultRouteTable params={"regionCode":"KR","vpcNo":"5068"}
2021/02/24 12:13:48 [INFO] getDefaultRouteTable response={"requestId":"d8d1bef3-f208-4114-9135-2629936cd8e0","returnCode":"0","returnMessage":"success","totalRows":2,"routeTableList":[{"rou
2021/02/24 12:14:01 [ERROR] resourceNcloudLbCreate error params={"regionCode":"KR","idleTimeout":30,"loadBalancerDescription":"tf test description","loadBalancerNetworkTypeCode":"PRIVATE","
cxwxz","throughputTypeCode":"SMALL","vpcNo":"5068","subnetNoList":["9047"]}, err=Status: 400 Bad Request, Body: {"responseError": {
  "returnCode": "908",
  "returnMessage": "Please check your input value : [80]. Vaild values : [HTTP, HTTPS]. location : protocolTypeCode"
}}
```

Terraform provider 만들기

DEVIEW 2021

Documentation

- /docs내의 Markdown파일 수정
- Doc Preview Tool 제공
- registry.terraform.io/tools/doc-preview
- tfplugindocs을 사용하여 자동화 가능



ncloud

Overview

NCLOUD DOCUMENTATION

Filter

ncloud provider

Resources

- ncloud_access_control_group
- ncloud_access_control_group_rule
- ncloud_auto_scaling_group
- ncloud_auto_scaling_policy
- ncloud_auto_scaling_schedule
- ncloud_block_storage
- ncloud_block_storage_snapshot
- ncloud_init_script
- ncloud_launch_configuration
- ncloud_lb
- ncloud_lb_listener
- ncloud_lb_target_group
- ncloud_lb_target_group_attachment
- ncloud_load_balancer
- ncloud_load_balancer_ssl_certificate
- ncloud_login_key
- ncloud_nas_volume
- ncloud_nat_gateway
- ncloud_network_acl
- ncloud_network_acl_rule

Resource: ncloud_vpc

Provides a VPC resource.

Example Usage

Basic Usage

```
resource "ncloud_vpc" "vpc" {
 ipv4_cidr_block = "10.0.0.0/16"
}

resource "ncloud_network_acl" "nacl" {
 vpc_no = ncloud_vpc.vpc.id
}
```

Argument Reference

The following arguments are supported:

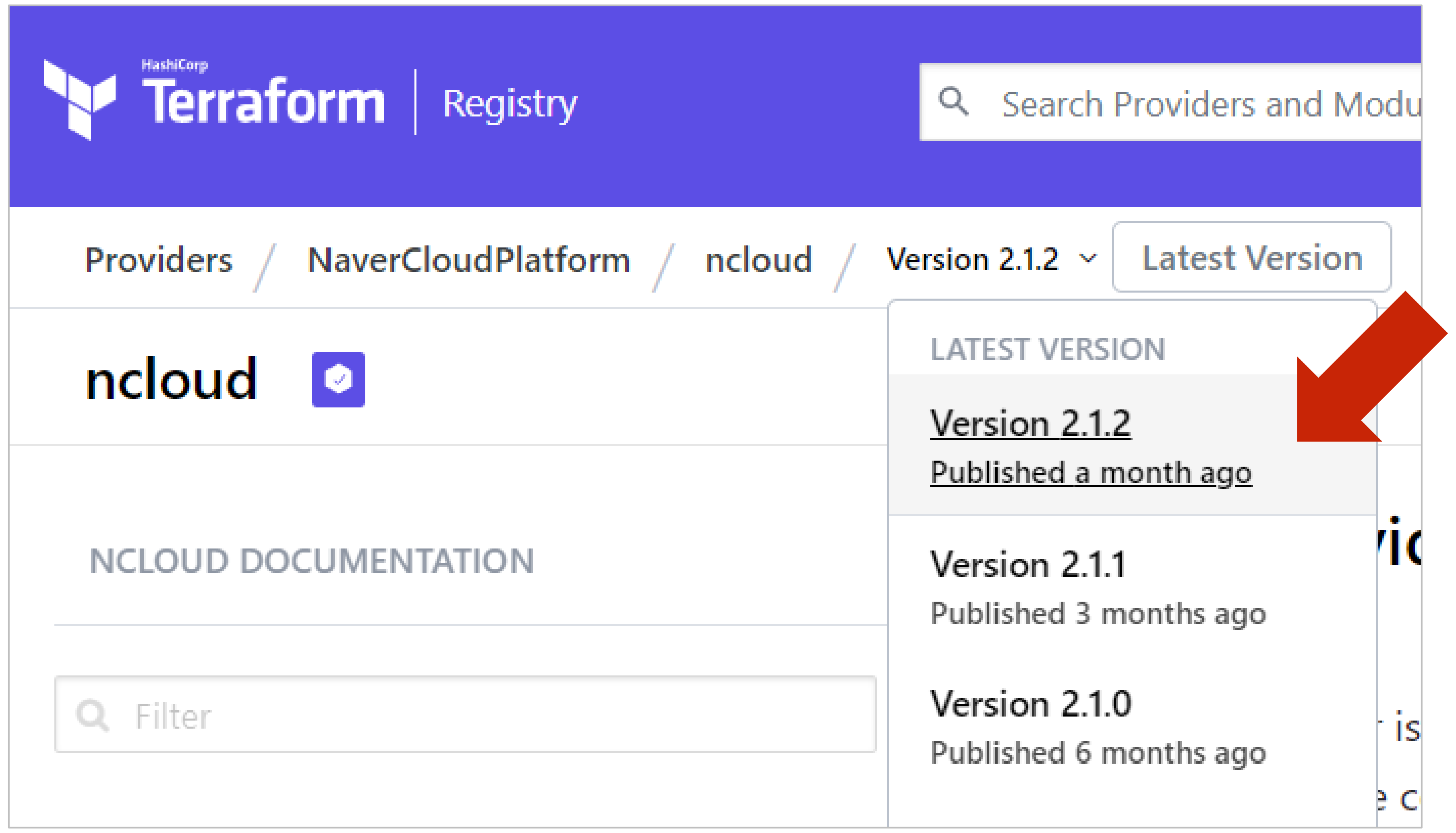
- `name` - (Optional) The name to create. If omitted, Terraform will assign a random, unique name.
- `ipv4_cidr_block` - (Required) The CIDR block of the VPC. The range must be between /16 and/28 within the private band (10.0.0/8,172.16.0.0/12,192.168.0.0/16).

https://www.terraform.io/docs/registry/providers/publishing.html

Terraform provider 만들기

Release

- Travis CI/CD 적용
- gorelease를 통해 바이너리 배포



Terraform Registry

Search Providers and Modu

Providers / NaverCloudPlatform / ncloud / Version 2.1.2 | Latest Version

ncloud

NCLOUD DOCUMENTATION

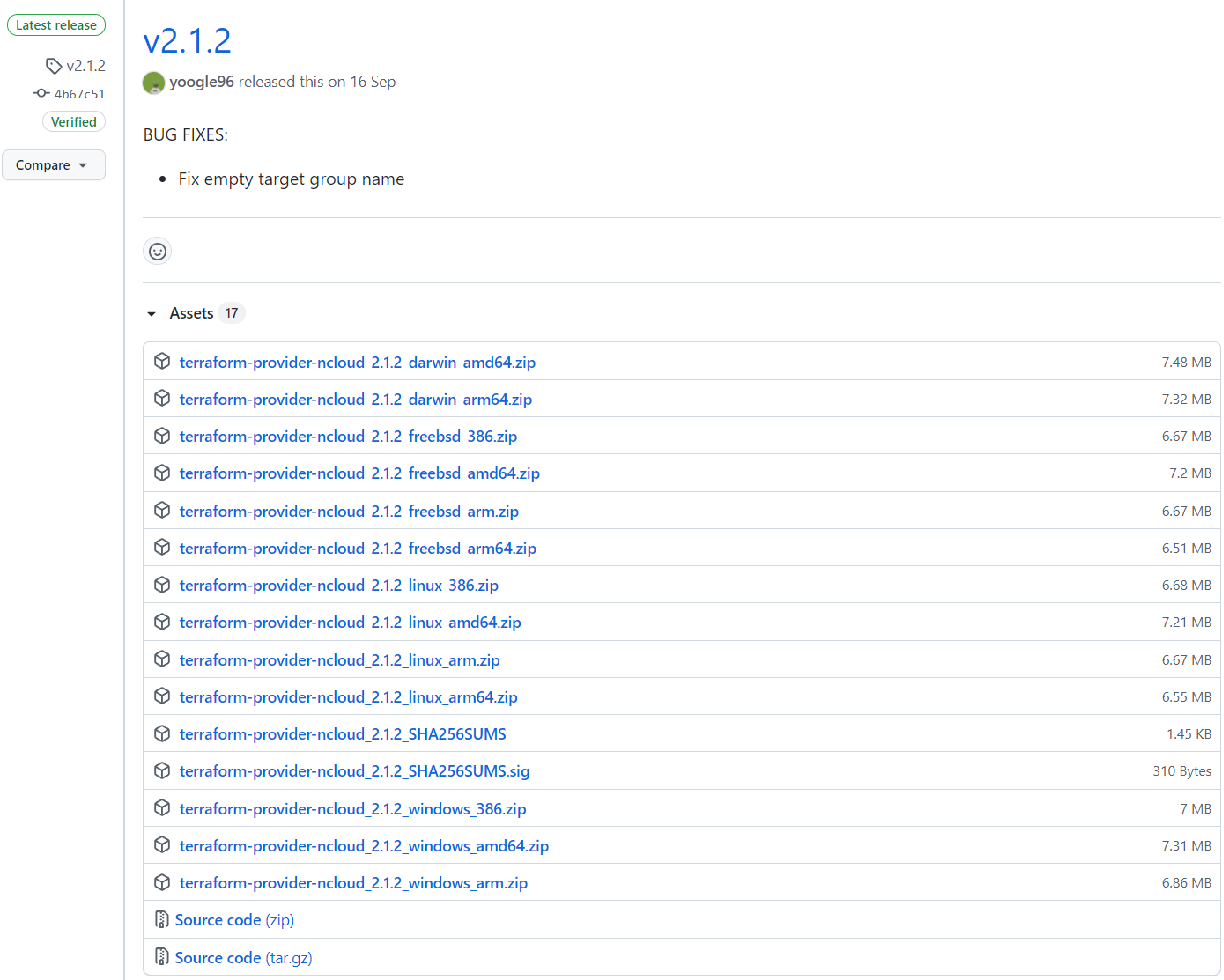
Filter

LATEST VERSION

Version 2.1.2
Published a month ago

Version 2.1.1
Published 3 months ago

Version 2.1.0
Published 6 months ago



Latest release

v2.1.2

4b67c51

Verified

Compare

v2.1.2

yoogle96 released this on 16 Sep

BUG FIXES:

- Fix empty target group name

Assets 17

terraform-provider-ncloud_2.1.2_darwin_amd64.zip	7.48 MB
terraform-provider-ncloud_2.1.2_darwin_arm64.zip	7.32 MB
terraform-provider-ncloud_2.1.2_freebsd_386.zip	6.67 MB
terraform-provider-ncloud_2.1.2_freebsd_amd64.zip	7.2 MB
terraform-provider-ncloud_2.1.2_freebsd_arm.zip	6.67 MB
terraform-provider-ncloud_2.1.2_freebsd_arm64.zip	6.51 MB
terraform-provider-ncloud_2.1.2_linux_386.zip	6.68 MB
terraform-provider-ncloud_2.1.2_linux_amd64.zip	7.21 MB
terraform-provider-ncloud_2.1.2_linux_arm.zip	6.67 MB
terraform-provider-ncloud_2.1.2_linux_arm64.zip	6.55 MB
terraform-provider-ncloud_2.1.2_SHA256SUMS	1.45 KB
terraform-provider-ncloud_2.1.2_SHA256SUMS.sig	310 Bytes
terraform-provider-ncloud_2.1.2_windows_386.zip	7 MB
terraform-provider-ncloud_2.1.2_windows_amd64.zip	7.31 MB
terraform-provider-ncloud_2.1.2_windows_arm.zip	6.86 MB
Source code (zip)	
Source code (tar.gz)	

https://www.terraform.io/docs/registry/providers/publishing.html

Terraform provider 만들기

정리하면

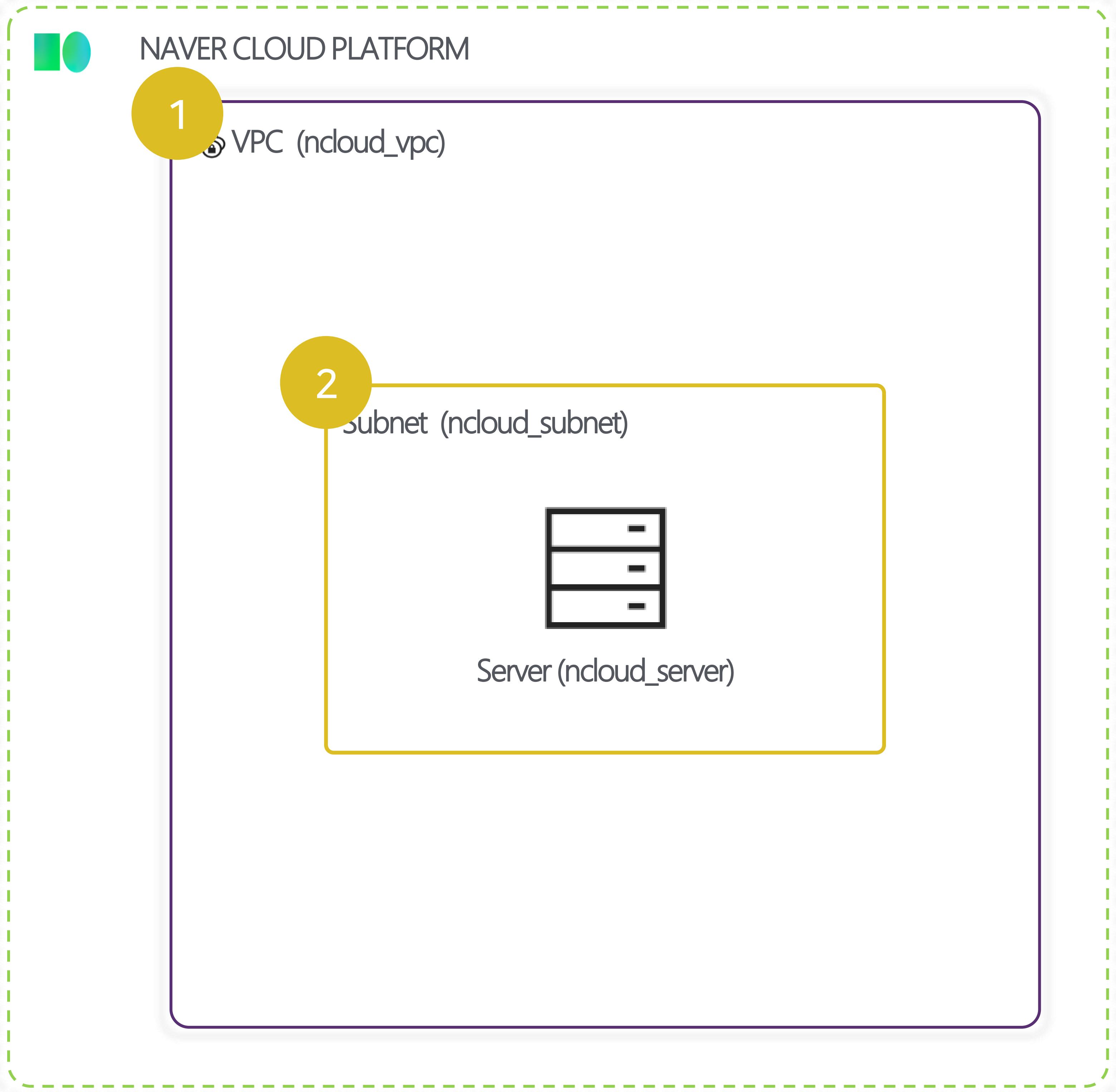
- GO Client library
- Provider 정의 (처음에만)
- Resource
  - Schema 정의
  - CRUD 구현
- Testing
- Debugging
- Documentation
- Release

4. Appendix & Tips

비동기 API 및 상태 처리

Resource graph

Terraform core에서는 Resource간의 종속성 그래프를 생성

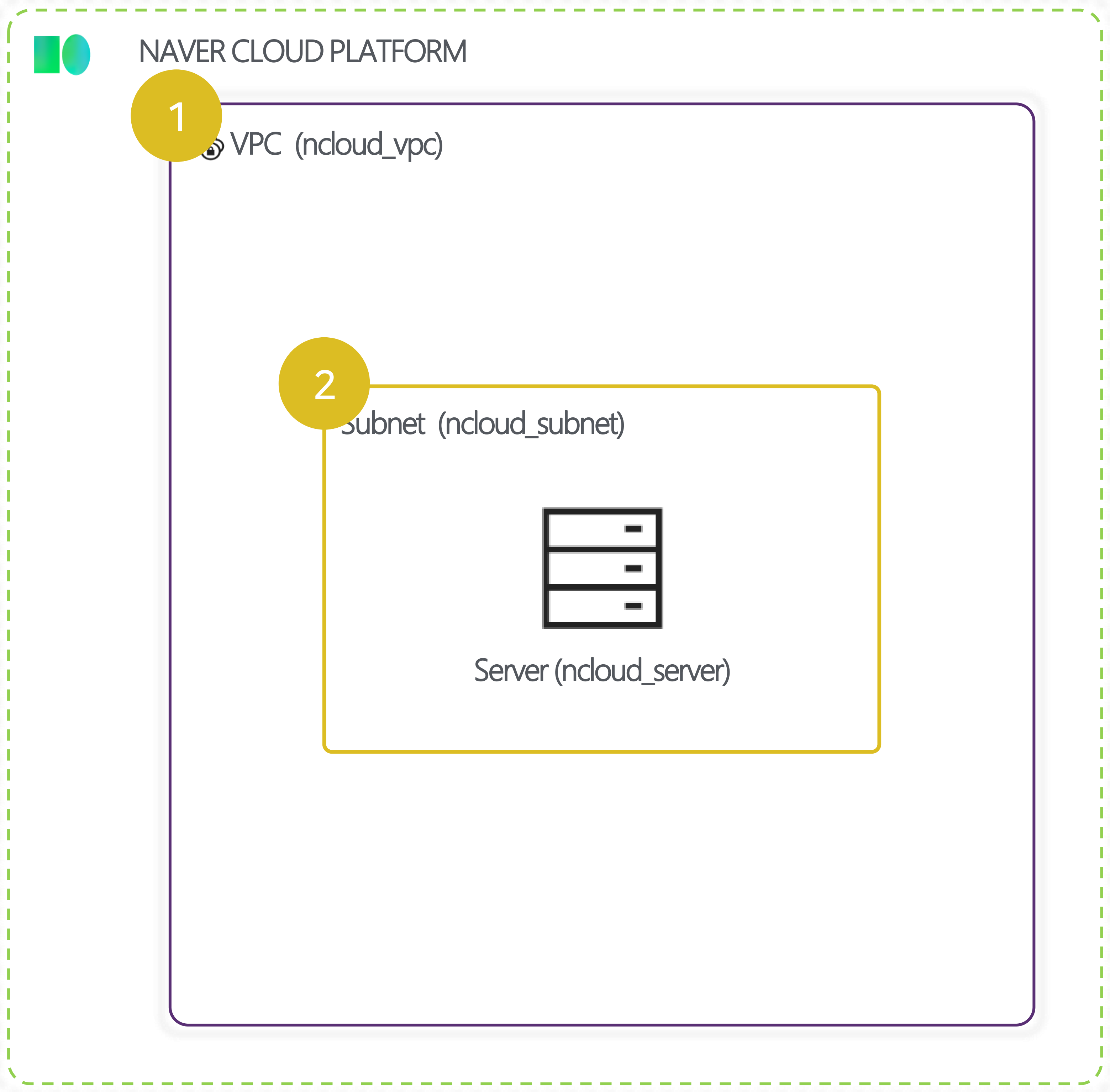


```
resource "ncloud_vpc" "vpc" {
  ipv4_cidr_block = "10.0.0.0/16"
}

resource "ncloud_subnet" "pub-sub" {
  vpc_no         = ncloud_vpc.vpc.id
  subnet         = cidrsubnet(ncloud_vpc.vpc.ipv4_cidr_block, 8, 1)
  zone           = "KR-2"
  network_acl_no = ncloud_vpc.vpc.default_network_acl_no
  subnet_type    = "PUBLIC"
}

resource "ncloud_server" "server" {
  subnet_no                 = ncloud_subnet.pub-sub.id
  name                      = "my-tf-server"
  server_image_product_code = "SW.VSVR.OS.LNX64.CNTOS.0703.B050"
```

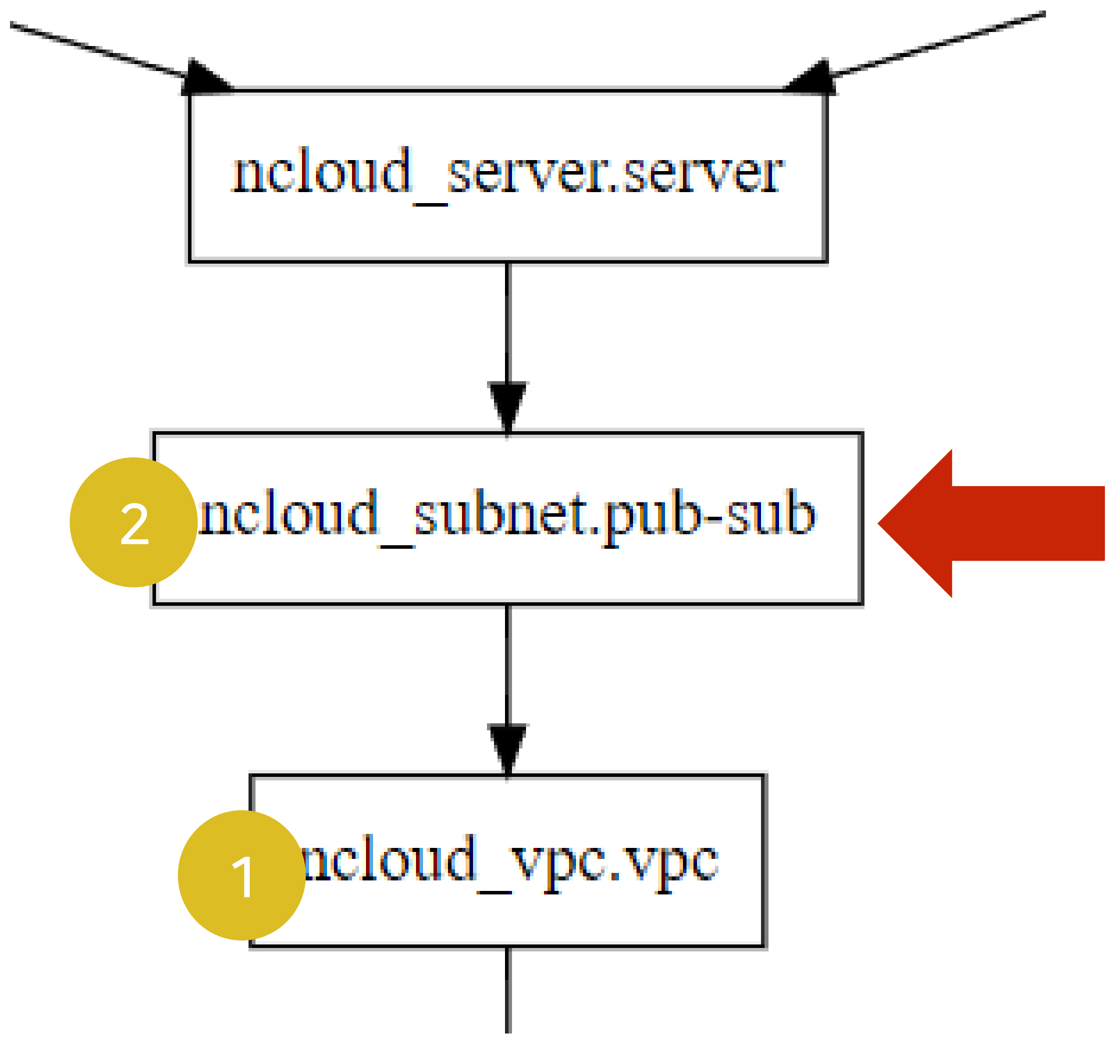
비동기 API 및 상태 처리

Resource graph

Terraform core에서는 Resource간의 종속성 그래프를 생성



```
$ terraform graph | dot -Tsvg > graph.svg
```



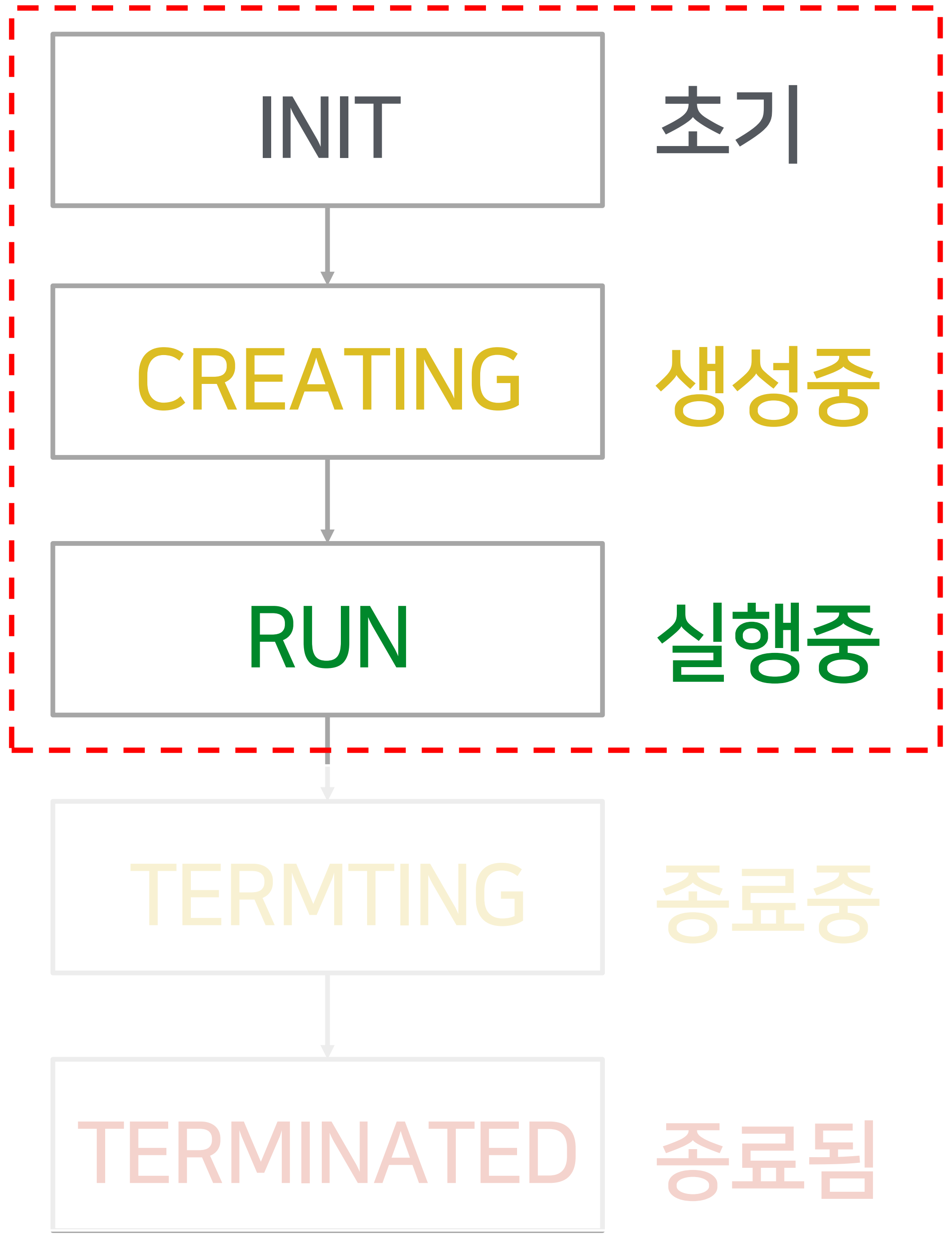
VPC가 완전히 생성 되기 전 Subnet 생성 불가

비동기 API 및 상태 처리

StateChangeConf

- Pending에서 Target상태가 될 때 까지 대기

VPC 인스턴스 생성 시



INIT 초기

CREATING 생성중

RUN 실행중

TERMTING 종료중

TERMINATED 종료됨

VPC상태가
RUN 상태가 될 때 까지 대기

```go
func resourceNcloudVpcCreate(ctx context.Context, d *schema.ResourceData, meta
interface{}) diag.Diagnostics {
    config := meta.(*ProviderConfig)

    reqParams := &vpc.CreateVpcRequest{
        RegionCode:    &config.RegionCode,
        VpcName:       ncloud.String(d.Get("name").(string)),
        Ipv4CidrBlock: ncloud.String(d.Get("ipv4_cidr_block").(string)),
    }

    resp, err := config.Client.vpc.V2Api.CreateVpc(ctx, reqParams)
    if err != nil {
        return diag.FromErr(err)
    }

    vpcInstance := resp.VpcList[0]
    d.SetId(*vpcInstance.VpcNo)
    log.Printf("[INFO] VPC ID: %s", d.Id())

    if err := waitForNcloudVpcCreation(ctx, config, d.Id()); err != nil {
        return diag.FromErr(err)
    }

    return resourceNcloudVpcRead(ctx, d, meta)
}
```

비동기 API 및 상태 처리

StateChangeConf

- Pending에서 Target상태가 될 때 까지 대기

```
func waitForNcloudVpcCreation(ctx context.Context, config *ProviderConfig, id string) error {
    stateConf := &resource.StateChangeConf{
        Pending: []string{"INIT", "CREATING"},
        Target:  []string{"RUN"},
        Refresh: func() (interface{}, string, error) {
            instance, err := getVpcInstance(config, id)
            return VpcCommonStateRefreshFunc(instance, err, "VpcStatus")
        },
        Timeout:    10 * time.Minute,
        Delay:      2 * time.Second,
        MinTimeout: 3 * time.Second,
    }

    if _, err := stateConf.WaitForStateContext(ctx); err != nil {
        return fmt.Errorf("Error waiting for VPC (%s) to become available: %s", id, err)
    }

    return nil
}
```

대기 할 State

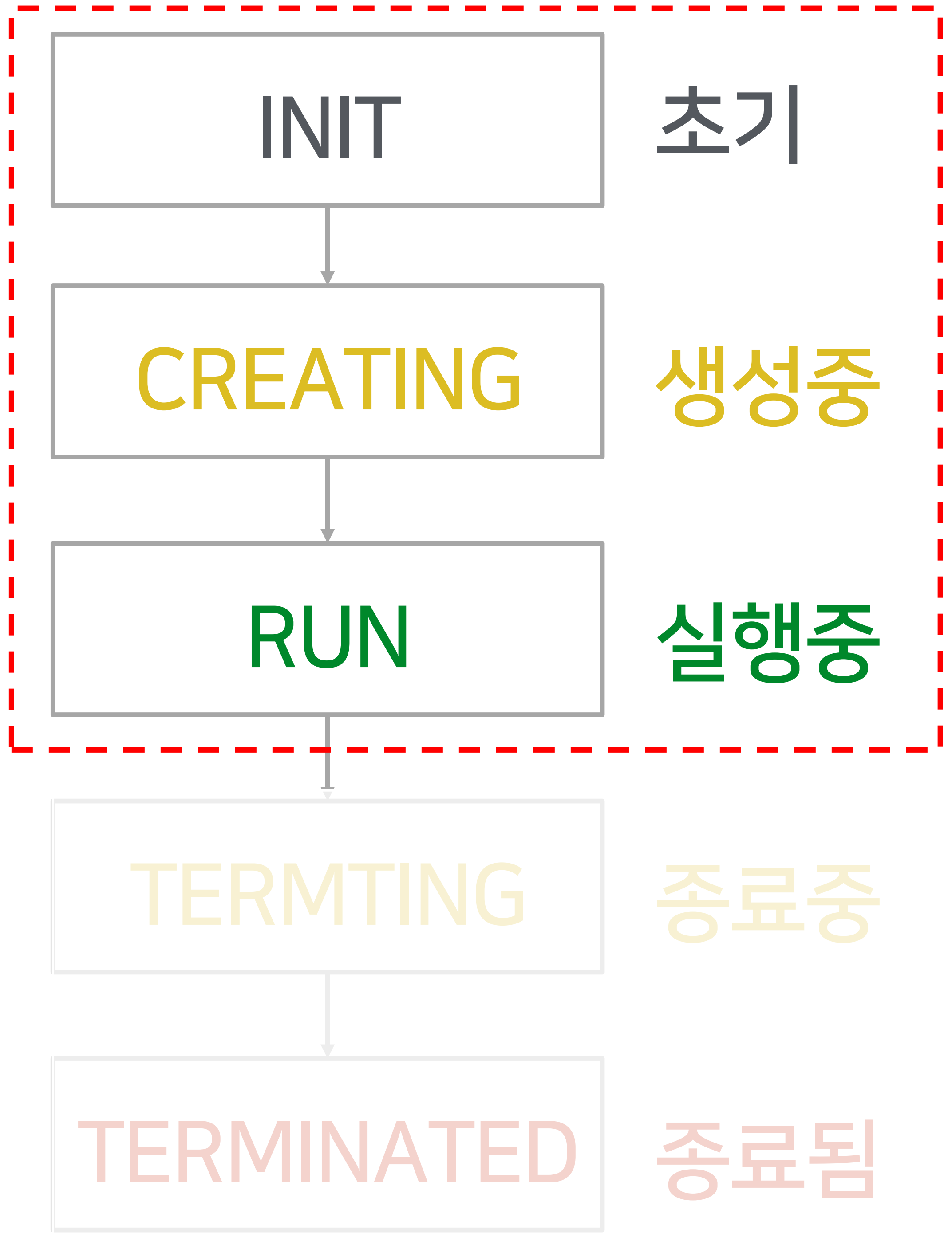
종료 시킬 State

상태 체크 로직

최대 대기하는 시간

해당 간격으로 상태 체크

VPC 인스턴스 flow



VPC 생성 시

(약 10초간 소요)

https://www.terraform.io/docs/extend/resources/retries-and-customizable-timeouts.html

동시성 이슈 처리

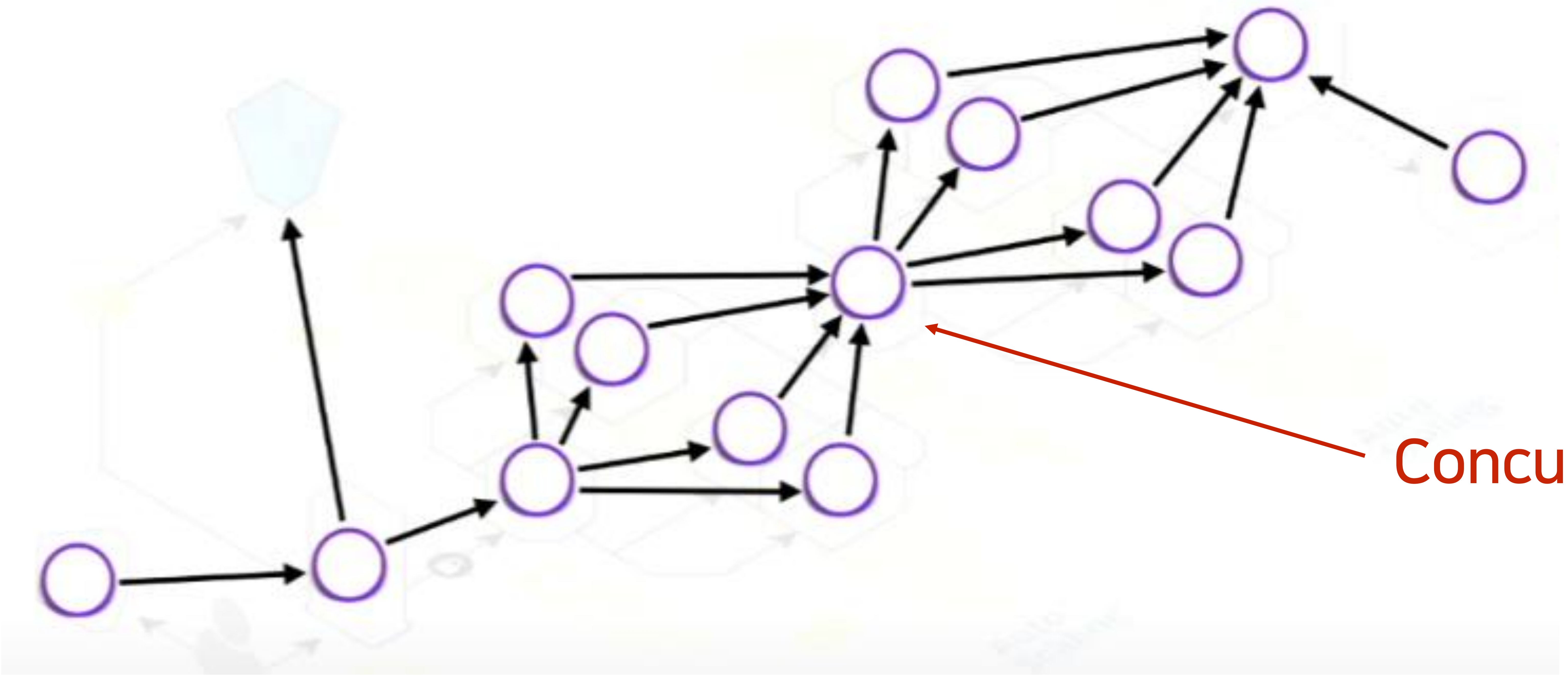
Parallel walk

- Terraform은 최대 10개의 노드를 병렬로 사용

  (-parallelism=n 을 통해 동시 노드 조절 가능 plan, apply, destroy)

- 하나의 인프라의 여러 요청이 동시에 들어올 때

예) 하나의 ACG 또는 Network ACL 에 여러 룰 들이 추가 될 때



Concurrency issue

동시성 이슈 처리

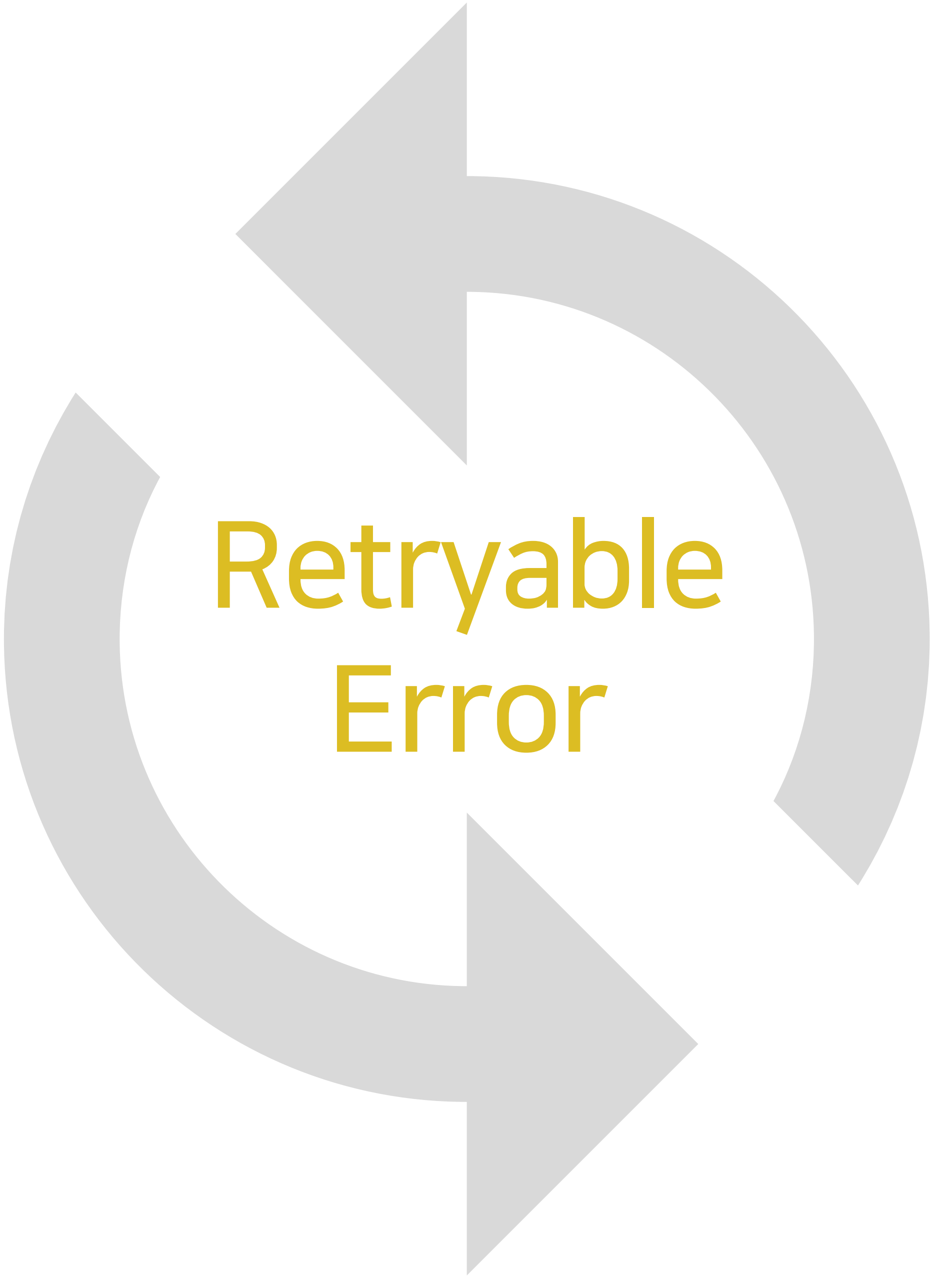
Retry

- 해당 자원에 접근이 일시적으로 어려울 경우 **재시도**

```
const ApiErrorNetworkAclCantAccessApropriate = "1011002"
/.../
err := resource.RetryContext(ctx, d.Timeout(schema.TimeoutCreate), func() *resource.RetryError {
        var err error
        /.../
        if err != nil {
            errBody, _ := GetCommonErrorBody(err)
            if errBody.ReturnCode == ApiErrorNetworkAclCantAccessApropriate {
                logErrorResponse("retry AddNetworkAclRule", err, reqParams)
                time.Sleep(time.Second * 5)
                return resource.RetryableError(err)
            }
            return resource.NonRetryableError(err)
        }
        return nil
    })
```

재 시도 (예상 가능한 에러)

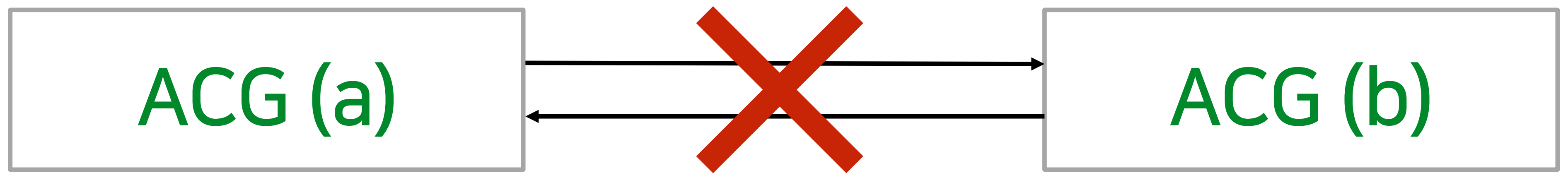
오류 발생 (재시도가 불가)



https://www.terraform.io/docs/extend/resources/retries-and-customizable-timeouts.html

순환참조(Circular dependecy) 이슈

두 Resource간 서로 참조하려고 할 때 Circular dependecy이슈 발생



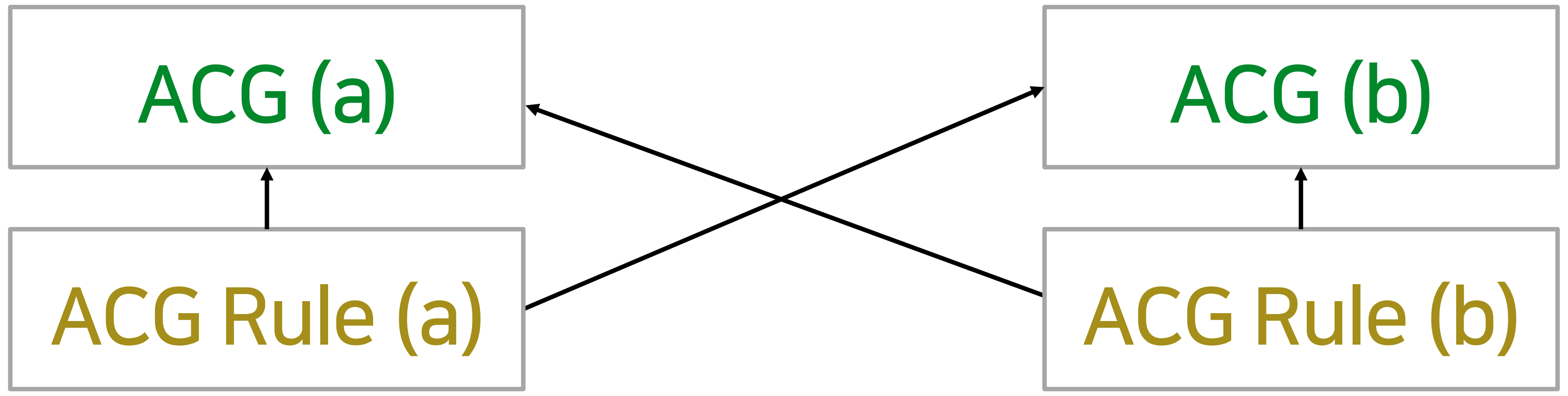
순환참조 리소스 생성 불가

```
resource "ncloud_access_control_group" "a" {
  vpc_no      = ncloud_vpc.vpc.id

  inbound {
    protocol                       = "TCP"
    port_range                     = "22"
    source_access_control_group_no = ncloud_access_control_group.b.id
  }
}
```

```
resource "ncloud_access_control_group" "b" {
  vpc_no      = ncloud_vpc.vpc.id

  inbound {
    protocol                       = "TCP"
    port_range                     = "22"
    source_access_control_group_no = ncloud_access_control_group.a.id
  }
}
```

순환참조(Circular dependecy) 이슈

리소스를 분리하여 순환참조 이슈 해결



```
resource "ncloud_access_control_group" "a" {
  vpc_no      = ncloud_vpc.vpc.id

resource "ncloud_access_control_group_rules" "acg-rule-a" {
  inbound {
    protocol                       = "TCP"
    port_range                     = "22"
    source_access_control_group_no = ncloud_access_control_group.b.id
  }
}
```

```
resource "ncloud_access_control_group" "b" {
  vpc_no      = ncloud_vpc.vpc.id

resource "ncloud_access_control_group_rules" "acg-rule-b" {
  inbound {
    protocol                       = "TCP"
    port_range                     = "22"
    source_access_control_group_no = ncloud_access_control_group.a.id
  }
}
```

Tip1. filter기능 제공

필터 기능을 제공하자

- API 단에서 지원하는 필터 기능은 제한 적
- 사용자에게 많은 속성의 필터링 제공

```
data "ncloud_server_product" "product" {
  server_image_product_code = "SW.VSVR.OS.LNX64.CNTOS.0703.B050"
  // Search by 'CentOS 7.3 (64-bit)' image vpc

  filter {
    name   = "product_code"
    values = ["SSD"]
    regex  = true
  }

  filter {
    name   = "cpu_count"
    values = ["2"]
  }

  filter {
    name   = "memory_size"
    values = ["8GB"]
  }
}
```

서버단에서 제공하는 기능

(API 에 따라 의존 적)

필터로 제공하는 기능

(모든 속성 가능)

Tip2. init() 사용

리소스 등록 시 init() 사용

- 협업 간 소스 충돌을 줄이고 편리하게 resource를 등록 가능

provider.go (old)

```
func Provider() *schema.Provider {
    return &schema.Provider{
        Schema:        schemaMap(),
        DataSourcesMap: map[string]*schema.Resource{
            "ncloud_vpc": dataSourceNcloudVpc(),
            "ncloud_subnet": dataSourceNcloudSubnet(),
            // ...
        },
        ResourcesMap: map[string]*schema.Resource{
            "ncloud_vpc": resourceNcloudVpc(),
            "ncloud_subnet": resourceNcloudSubnet(),
            // ...
        },
        ConfigureFunc:  providerConfigure,
    }
}
```

→

provider.go (new)

```
func Provider() *schema.Provider {
    return &schema.Provider{
        Schema:         schemaMap(),
        DataSourcesMap: DataSourcesMap(),
        ResourcesMap:   ResourcesMap(),
        ConfigureFunc:  providerConfigure,
    }
}
```

resource_ncloud_vpc.go

```
func init() {
    RegisterResource("ncloud_vpc", resourceNcloudVpc())
}

func resourceNcloudVpc() *schema.Resource {
    return &schema.Resource{
        CreateContext: resourceNcloudVpcCreate,
        ReadContext:   resourceNcloudVpcRead,
        UpdateContext: resourceNcloudVpcUpdate,
        DeleteContext: resourceNcloudVpcDelete,
        Importer: &schema.ResourceImporter{
            State: schema.ImportStatePassthrough,
        },
        Schema: map[string]*schema.Schema{…},
    }
}
```

Tip3. 공통 함수를 재사용 하자

재사용 하기 좋은 함수들

- Marshall & Unmarshall functions
- Flatten 함수들 (API Model -> Resource schemas)
- Get functions
  (Resource와 DataSource, StateChangeConf 에서 사용)

Tip4. Go context

Context 구현

context.**Context**를 GO SDK에 전달하여 User cancellation 발생 시

불필요한 요청을 중단

resource_ncloud_vpc.go

```go
func resourceNcloudVpc() *schema.Resource {
    return &schema.Resource{
        CreateContext: resourceNcloudVpcCreate,
        ReadContext:   resourceNcloudVpcRead,
        UpdateContext: resourceNcloudVpcUpdate,
        DeleteContext: resourceNcloudVpcDelete,
        Importer: &schema.ResourceImporter{
            State: schema.ImportStatePassthrough,
        },
        Schema: map[string]*schema.Schema{…},
    }
}
```

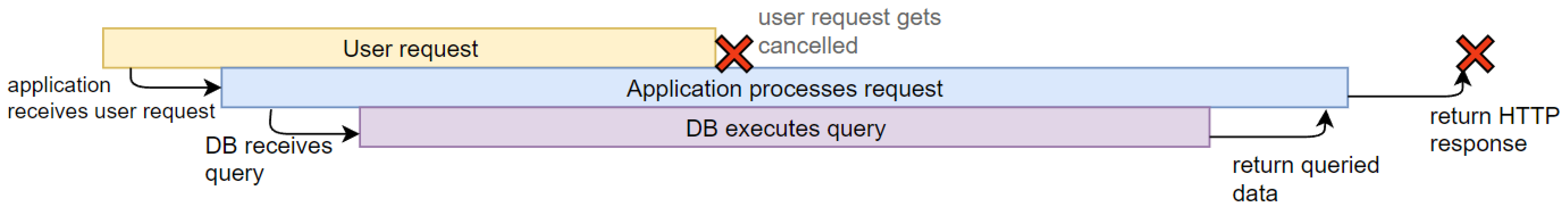
```go
func resourceNcloudVpcCreate(ctx context.Context, d *schema.ResourceData, meta interface{}) diag.Diagnostics {
    config := meta.(*ProviderConfig)

    reqParams := &vpc.CreateVpcRequest{
        RegionCode:    &config.RegionCode,
        VpcName:       ncloud.String(d.Get("name").(string)),
        Ipv4CidrBlock: ncloud.String(d.Get("ipv4_cidr_block").(string)),
    }

    resp, err := config.Client.vpc.V2Api.CreateVpc(ctx, reqParams)
    if err != nil {
        return diag.FromErr(err)
    }

    vpcInstance := resp.VpcList[0]
    d.SetId(*vpcInstance.VpcNo)

    return resourceNcloudVpcRead(ctx, d, meta)
}
```

Tip4. Go context

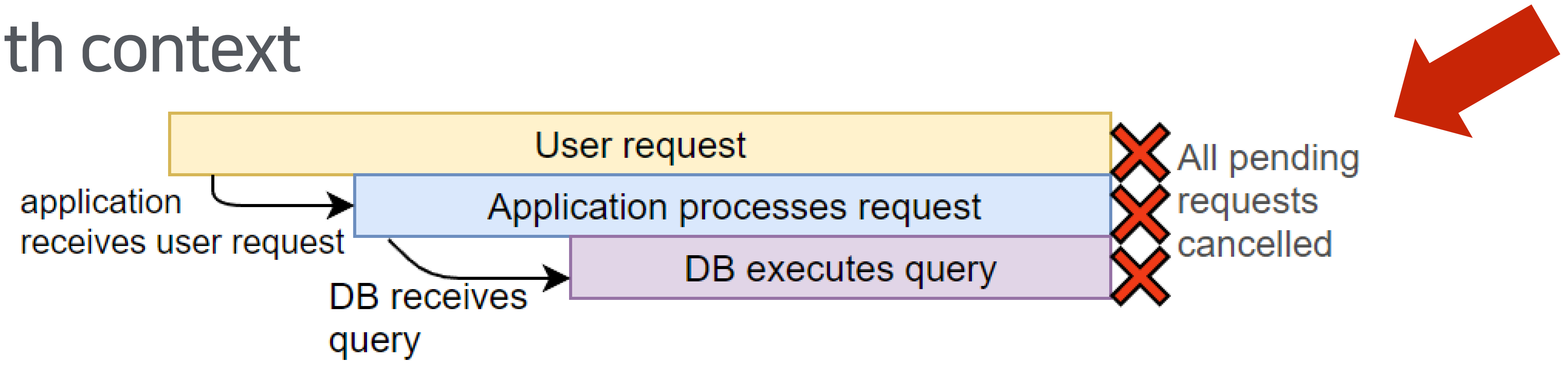
Context 구현

context.Context를 GO SDK에 전달하여 User cancellation 발생 시
불필요한 요청을 중단

Without context


User request
user request gets cancelled
application receives user request
Application processes request
DB receives query
DB executes query
return queried data
return HTTP response

With context


User request
application receives user request
Application processes request
DB receives query
DB executes query
All pending requests cancelled

https://www.sohamkamani.com/golang/context-cancellation-and-values/

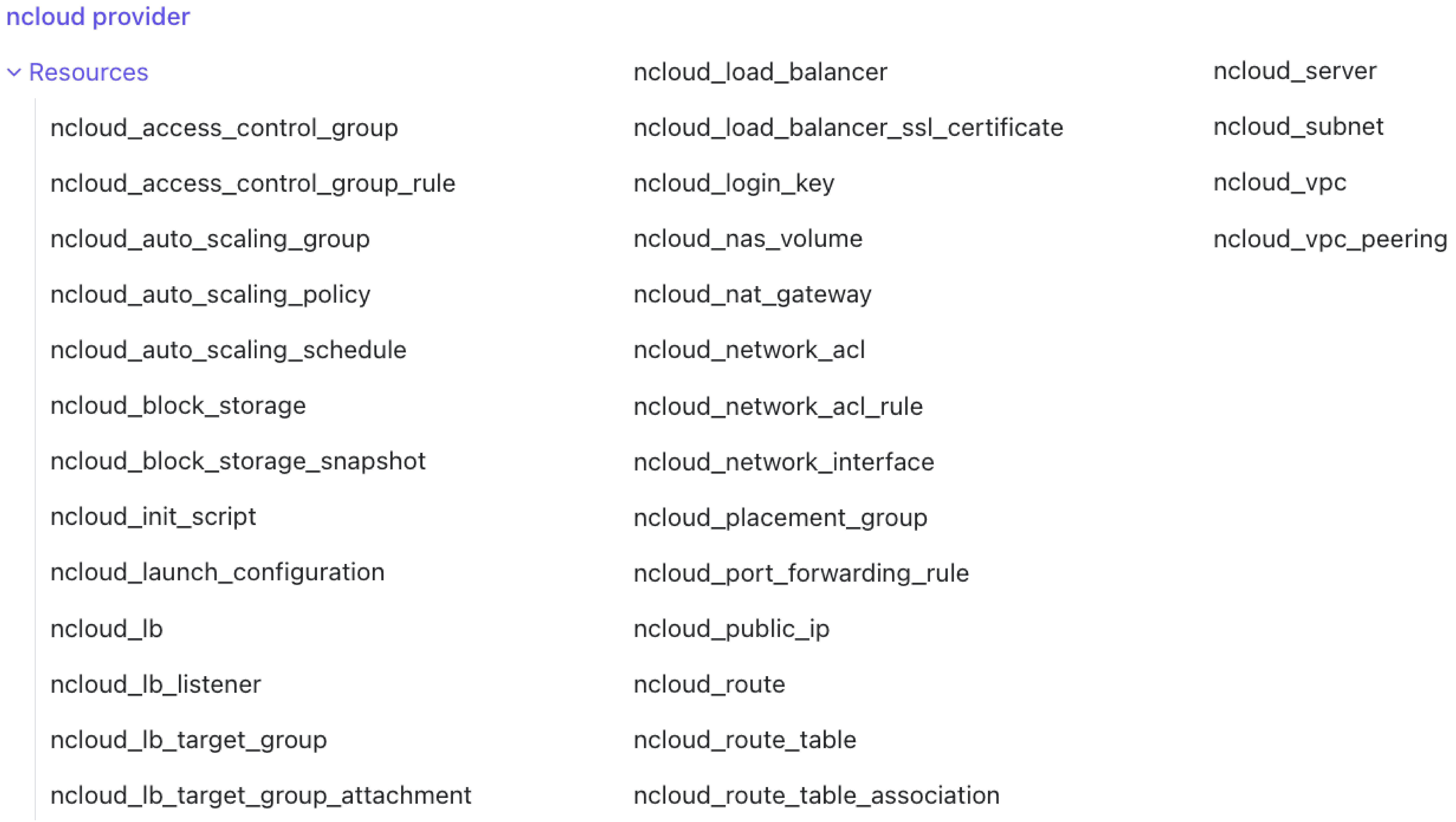
5. Summary

Contributor

- 유성덕
- 임근대
- 유의선
- 김상규

Thanks for

- 배영수
- 김준희
- 최왕용

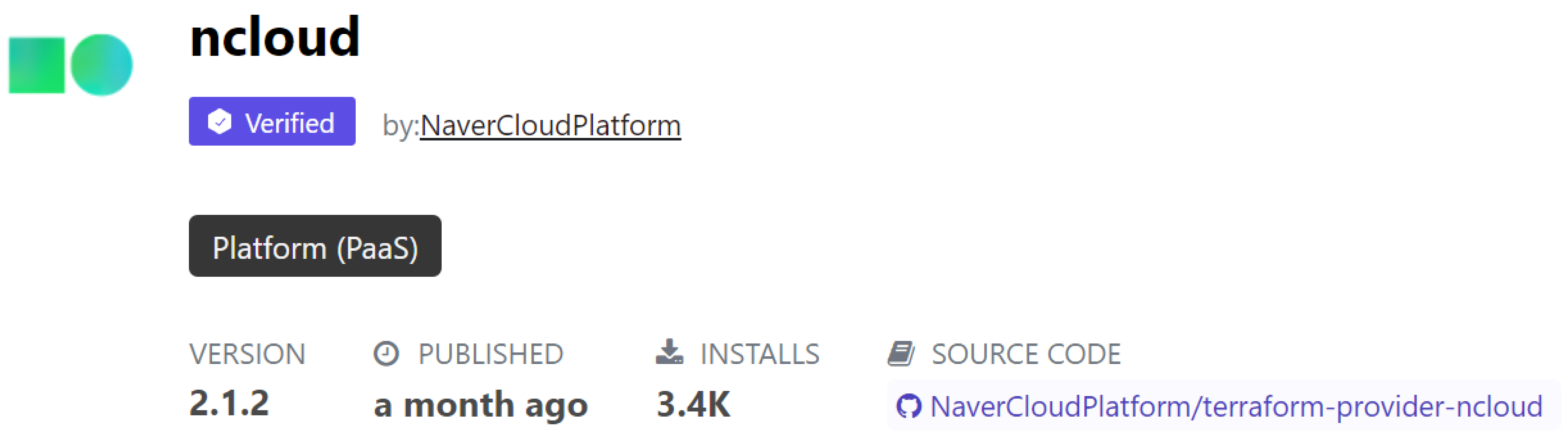


ncloud provider

Resources

ncloud_access_control_group
ncloud_access_control_group_rule
ncloud_auto_scaling_group
ncloud_auto_scaling_policy
ncloud_auto_scaling_schedule
ncloud_block_storage
ncloud_block_storage_snapshot
ncloud_init_script
ncloud_launch_configuration
ncloud_lb
ncloud_lb_listener
ncloud_lb_target_group
ncloud_lb_target_group_attachment
ncloud_load_balancer
ncloud_load_balancer_ssl_certificate
ncloud_login_key
ncloud_nas_volume
ncloud_nat_gateway
ncloud_network_acl
ncloud_network_acl_rule
ncloud_network_interface
ncloud_placement_group
ncloud_port_forwarding_rule
ncloud_public_ip
ncloud_route
ncloud_route_table
ncloud_route_table_association
ncloud_server
ncloud_subnet
ncloud_vpc
ncloud_vpc_peering

Join contributor



ncloud

Verified by:NaverCloudPlatform

Platform (PaaS)

VERSION 2.1.2

PUBLISHED a month ago

INSTALLS 3.4K

SOURCE CODE NaverCloudPlatform/terraform-provider-ncloud

https://github.com/NaverCloudPlatform/terraform-provider-ncloud

무엇이든? 누구든?

HTTP(S) API 가 있다면

Terraform provider가 될 수 있다

Thank You