



# Terraform

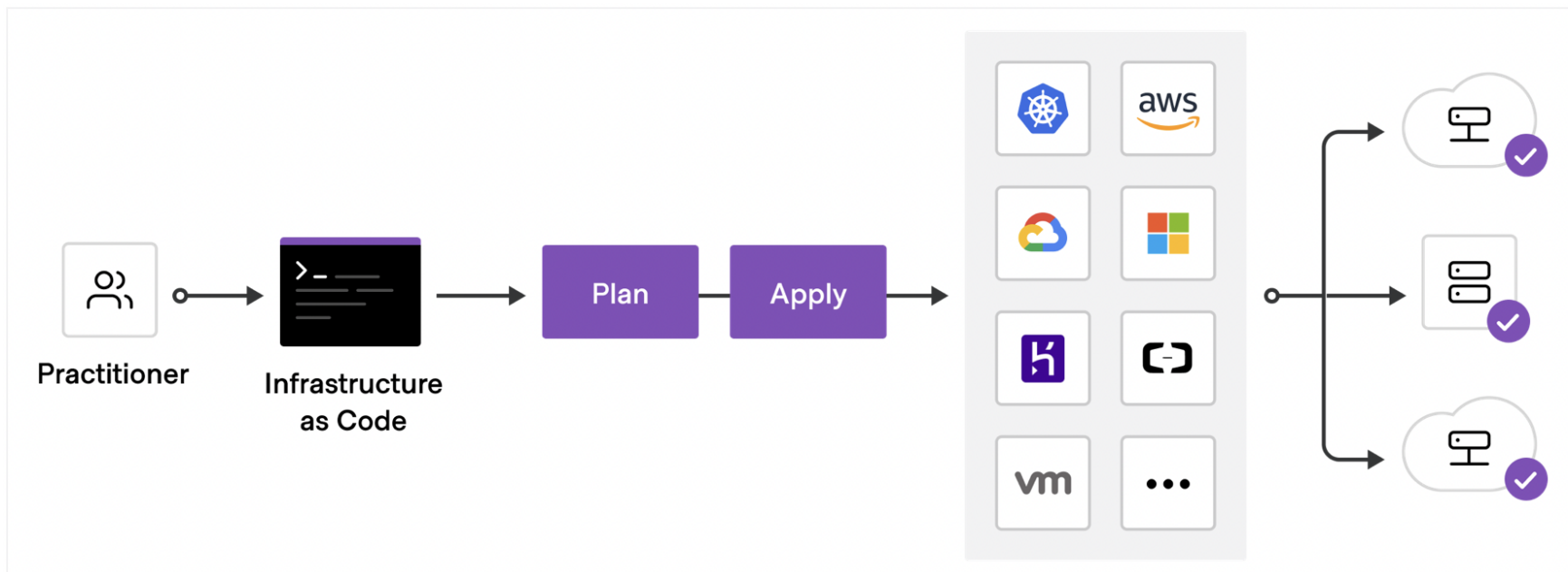
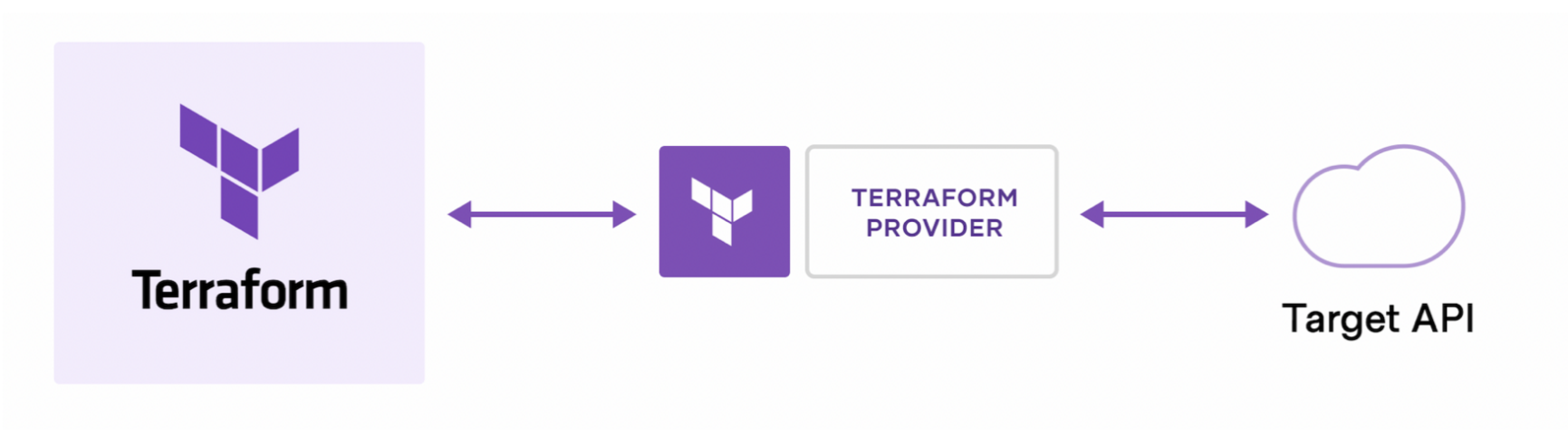
## 1.테라폼이란?

HashiCorp Terraform is an infrastructure as code tool that lets you define both cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share.

- Cloud나 On-prem 환경 상관 없이 사람이 읽기 쉬운 방법으로 인프라를 **코드**로써 관리하는 도구
- 여러 클라우드 제공 업체에 걸쳐있는 매우 큰 인프라를 관리 할 수 있도록 한다.
- 컴퓨팅, 저장소, 네트워크 뿐 아니라 DNS 설정 과 같이 인프라 구축 및 프로비저닝이 가능하도록 한다.

## 2. 작동 원리

- Provider와 API를 통해서 다른 클라우드 플랫폼이나 다른 서비스에 리소스를 생성하고 관리한다.




- terraform 의 `main.tf` 의 코드를 작성하고 apply
- 호환하는 여러 API를 통해서 스토리지, 네트워크, 도커 등 실행 및 관리가 가능.

## 3. 테라폼 설치

### Install Terraform | Terraform - HashiCorp Learn

To use Terraform you will need to install it. HashiCorp distributes Terraform as a binary package. You can also install Terraform using popular package managers. Retrieve the terraform binary by downloading a pre-compiled binary or compiling it from source. To install Terraform, find the appropriate

 <https://learn.hashicorp.com/tutorials/terraform/install-cli?in=terraform/docker-get-started>

 HashiCorp Learn

Explore our tutorials  
to automate your  
workflows



## 4. 예시 코드

Terraform *configuration 파일*

- `main.tf`

```
terraform {
  required_providers {
    docker = {
      source = "kreuzwerker/docker"
      version = "~> 2.13.0"
    }
  }
}

provider "docker" {}

resource "docker_image" "nginx" {
  name          = "nginx:latest"
  keep_locally = false
}

resource "docker_container" "nginx" {
  image = docker_image.nginx.latest
  name  = "tutorial"
  ports {
    internal = 80
    external = 8000
  }
}
```

### 4.1 terraform 블록

- `terraform {}` 블록은 Terraform 세팅, 프로비저닝할 `required_providers` 를 포함
  - `source` 속성은 hostname이나 namespace, type을 명시함.
  - 보통 Terraform Registry 에서 Provider 이미지를 가져옴
  - 해당 예제에서는 `registry.terraform.io/kreuzwerker/docker` 를 활용
- `required_providers` 블록에는 provider의 버전을 명시
  - 명시하지 않을 경우, 자동으로 recent 버전을 가져옴.
  - 호환성을 위해 명시하는 것을 권장

### 4.2 Providers 블록

- Terraform에서 생성하고 관리할 특정 provider 를 명시
- 여러개의 provider를 사용할 수 있음
  - 예) docker ID를 k8s에 넘겨주는 방식으로 사용 가능

### 4.3 Resource 블록

- infra의 컴포넌트에 대한 정보 블록, 도커와 같은 물리적 자원이나 heroku와 같은 논리적 자원 모두 가능
- Provider에 따른 resource 타입 및 이름을 명시함
- 2개의 string 타입을 명시 : 리소스 타입 / 리소스 이름
- 자원의 리소스 : 머신 크기, 디스크 이미지 이름, VPC ID등 Argument에 대한 정보가 들어감

- [providers reference](#) 참고

## 5. Terraform CLI

### 5.1 terraform 시작 및 유효성 검사

```
terraform init
```

- main.tf가 있는 폴더에서 테라폼 시작
- `.terraform` 폴더 : 설치한 provider
- `.terraform.lock.hcl` 파일 : 설치한 provider의 버전 명시

```
terraform fmt
```

- terraform의 포매팅 기능

```
terraform validate
```

- terraform config 파일의 유효성 검사

### 5.2 terraform 인프라 생성

#### terraform apply

- infrastructure 생성
- `terraform.tfstate` 파일에 ID나 속성값을 기록, 해당 파일을 이용하여 create 하거나 destroy
  - 따라서, 해당 파일을 securely하게 보관하거나 접근 제한을 두어 안전하게 유지해야 함

## 6. Terraform Input & Output

### Terraform Input

- terraform 에서는 필요한 내용을 변수로 생성하여 flexible하게 설정 가능하다.
- `variable.tf`
  - 참고 : terraform은 `.tf` 확장자의 파일을 모두 읽어오기 때문에, 파일명은 상관 없음.

```
variable "container_name" {
  description = "Value of the name for the Docker container"
  type        = string
  default     = "ExampleNginxContainer"
}
```

- 동적 변수로 할 이름을 `var.{설정할 변수명}` 으로 설정
- docker 이미지 명 : `docker_container.nginx`

```
resource "docker_container" "nginx" {
  image = docker_image.nginx.latest
- name  = "tutorial"
+ name  = var.container_name
  ports {
    internal = 80
    external = 8080
  }
}
```

- CLI 실행

```
$ terraform apply -var "container_name=YetAnotherName"
```

## Terraform Output

- terraform output 쿼리를 통해서 출력 결과를 볼 수 있다.
- terraform 설정 파일이 있는 폴더에 output.tf 파일 기록

```
output "container_id" {
  description = "ID of the Docker container"
  value       = docker_container.nginx.id
}

output "image_id" {
  description = "ID of the Docker image"
  value       = docker_image.nginx.id
}
```