Public Cloud Deployment with Terraform

Automating Infrastructure as Code (IAC)

UNIVERSITY OF OREGON

NSRC
Network Startup Resource Center

Deploying Infrastructure in AWS

Computing

- Instances: EC2 (Elastic Compute Cloud)
- Functions: AWS Lambda
- Containers: ECS (Elastic Container Service)
  EKS (Elastic Kubernetes Service)

Data storage

- Databases: RDS
- Objects: S3

plus Networks, ACLs, permissions …etc etc

Infrastructure as Code (IAC)

- Desired configuration defined in **text files**
- **Declarative**: define what you want the final system to look like, not the steps needed to get there
- Text files can be version controlled, diff'd, tested, rolled back
- Replaces "click-ops"

IAC benefits

- Manage infrastructure across providers
- Track and document infrastructure
- Automate changes
- Standardize configurations (repeatable)
- Versioning and Recovery
- Collaboration

Cloud Provider's own IAC services

AWS has *Cloud Formation* and *CDK*
Azure has *Azure Resource Manager*
Google has *Deployment Manager*
…

Very good support for all the resource types in that particular cloud.

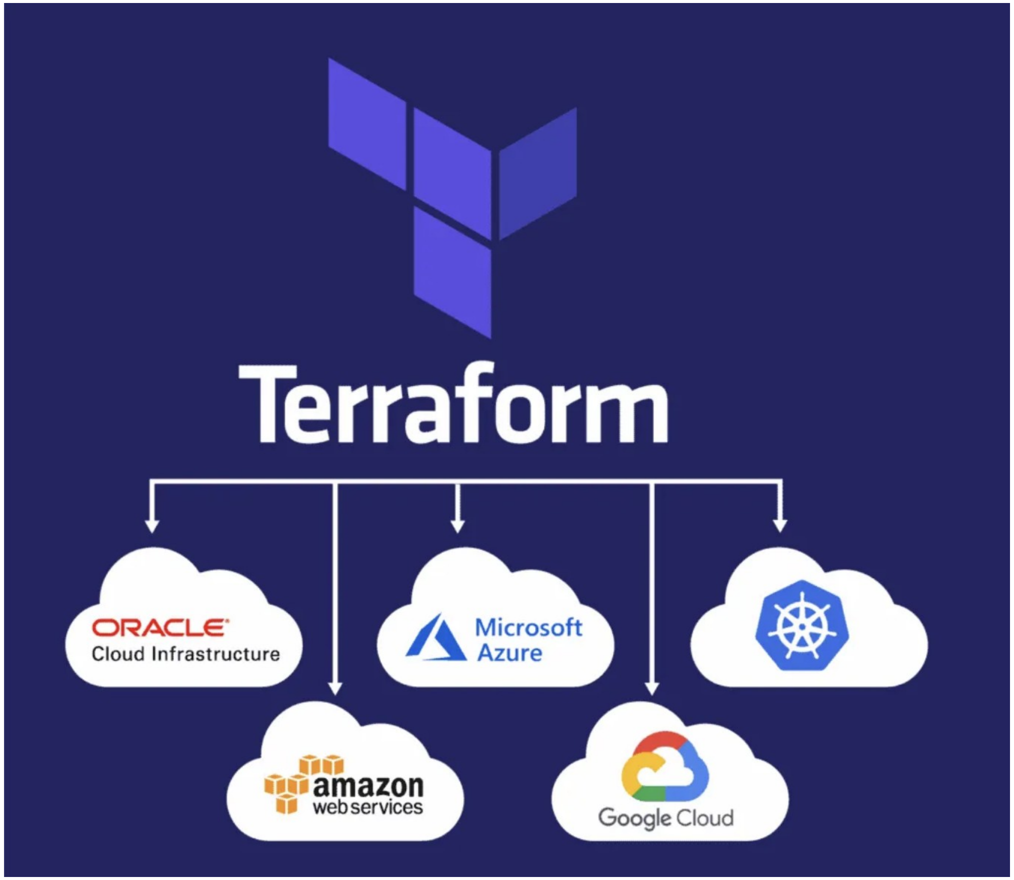
But tied to that one cloud.

What is Terraform?

- IAC tool to define resources in human-readable form
  - Version control , reuse and share this code

- Provision and manage infrastructure lifecycle consistently

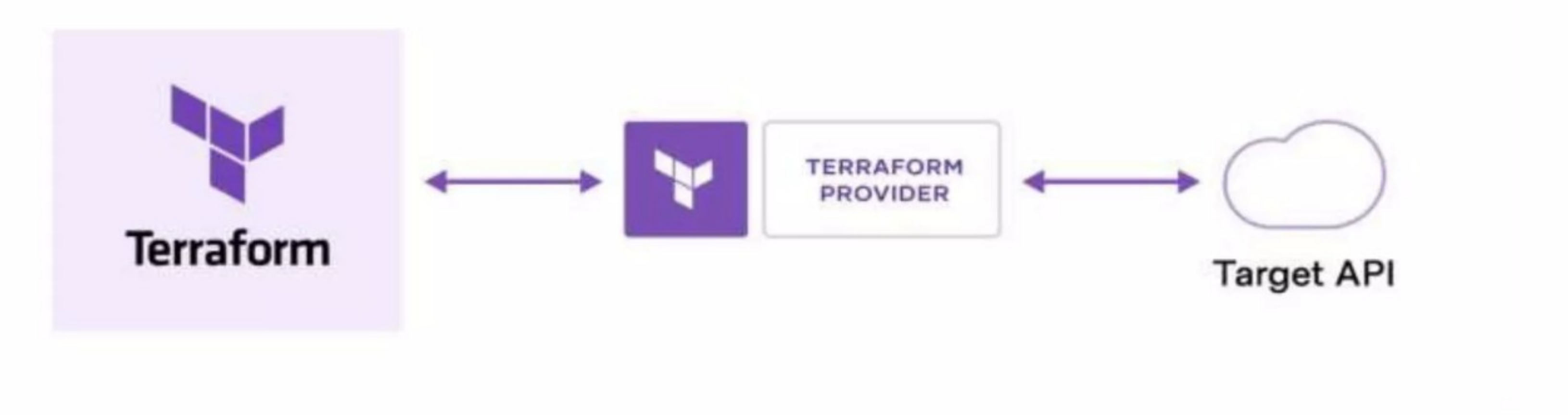
- Interacts with cloud platforms APIs via ”providers”



Terraform

- How does it work?
- Write: define resources in code
- Plan: execution plan describing changes
- Apply: execute the changes

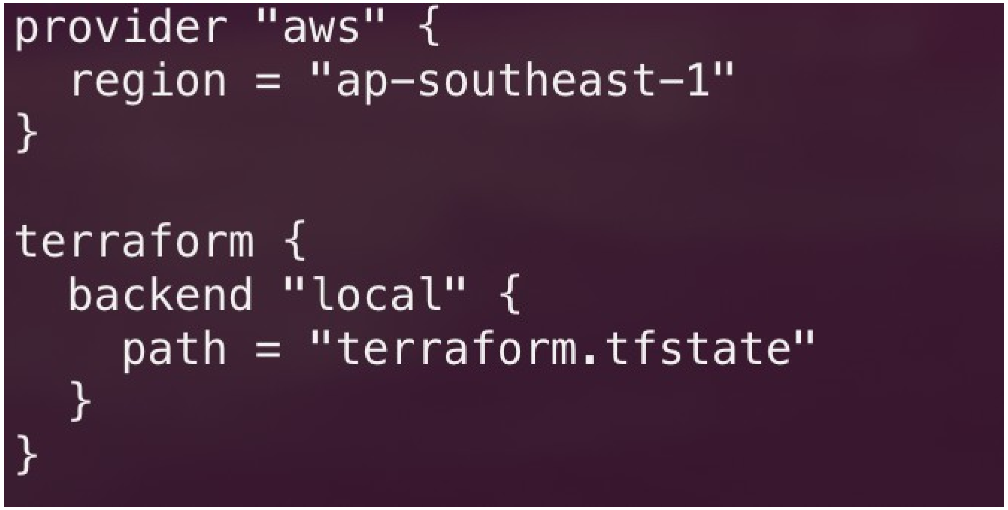
  create, change, destroy resources
- Show: show state of managed resources



HCL

Hashicorp Configuration Language

- designed for infrastructure automation
- syntax similar to JSON
- configuration language for Terraform
- also for other Hashicorp tools (Vault, Consul)



```
provider "aws" {
  region = "ap-southeast-1"
}

terraform {
  backend "local" {
    path = "terraform.tfstate"
  }
}
```

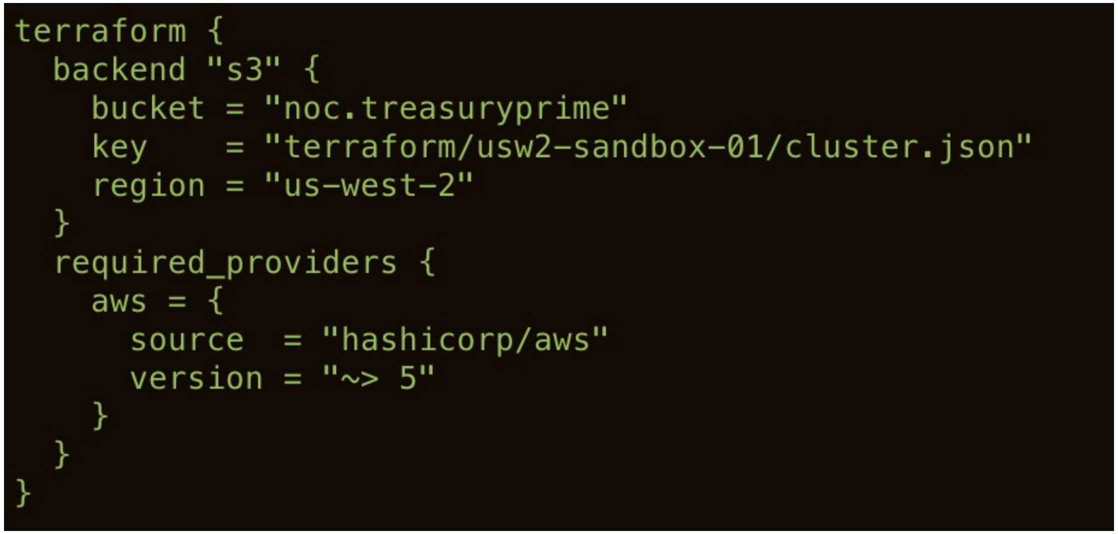
What is Terraform State Management?

Terraform.tfstate file

- Mapping real-world resources to configuration code (so they can be modified or deleted later)
- Keeping track of metadata and dependencies
- It is a database!!
- The state can be stored locally or remotely
  - Local storage only makes sense for testing
  - Best practice: always use remote storage in production

https://developer.hashicorp.com/terraform/language/state

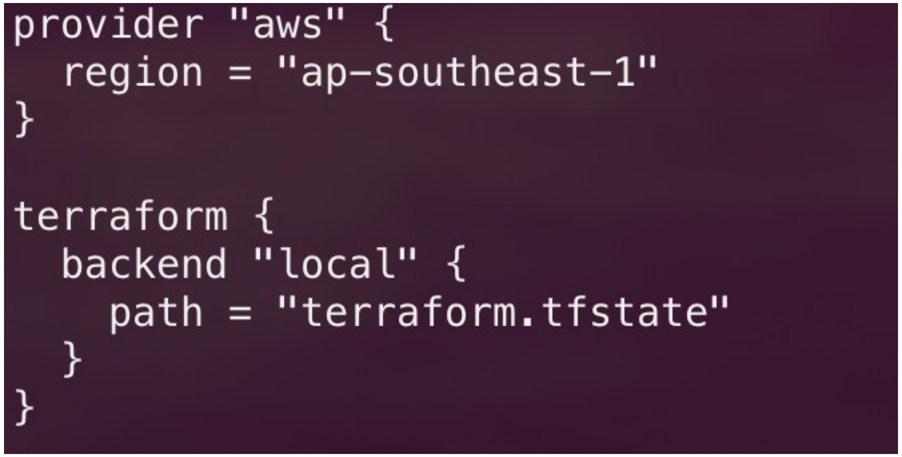
Terraform State



```
terraform {
  backend "s3" {
    bucket = "noc.treasuryprime"
    key    = "terraform/usw2-sandbox-01/cluster.json"
    region = "us-west-2"
  }
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5"
    }
  }
}
```

State stored remotely

State stored locally



```
provider "aws" {
  region = "ap-southeast-1"
}

terraform {
  backend "local" {
    path = "terraform.tfstate"
  }
}
```

What to be careful about with Terraform

State Management files

- Do not use local storage in production
- Back up state files
- Be careful about resource dependencies

What to be careful about with Terraform (II)

Terraform code

- Use variables and parameters
  - Do not hard-code values!
  - Do not put secrets or credentials in code!
- Write reusable modules, do not repeat code
- Keep code under version control (git, Github, GitLab)

What to be careful about with Terraform (III)

Change management

- Carefully inspect “terraform plan” output data before applying changes
- Do not make manual changes to provisioned resources
- Keep informed about changes in providers APIs

Terraform License

Not OSS, but free to use in almost all cases

You don't need to worry about it unless you're using Terraform to sell a service that competes with Terraform itself

There is a fork called *OpenTofu* - but Terraform has enough rough edges as it is, without giving yourself more headaches!