Note Files on Github/myaccount IaC

1

Infrastructure As Code (IAC)

provisionning infrastructure through sw to achieve consistent predictible environment

Core Concept

1. Defined in Code
   - config files
2. Stored in Source Control
   - ex: github
3. Declarative or Imperative

Ex. Make sandwich (Taco)

Declarative	Imperative
I want something of type food and subtype taco	
# Make me Taco	# Make me Taco
food (1) taco (2) "Bean-taco" {	get shell
ingredients ["beans", "cheese", "lettuce", "salsa"]	get beans
↑ config "predefined nourriture"	get cheese
}	get lettuce
	⋮
	put beans in shell
	⋮

what

how

(4) Idempotent and consistency

(5) Pushing / pulling

Idempotent

If a friend asks me for a sandwich, I give her a one, if she asks again at the same time I would say no because I know she has a one (I know her state (idempotent))

Terraform is a push type environment i.e. when I know that my friends needs a sandwich, I push her a one. on the contrary pull system means, once she needs a sandwich, she pulls a one.

Scanned with CamScanner

IaC Benefits:..

- Automated deployment
- Consistent environments (same each time)
- Repeatable process
- Reusable Components

  ex deploying DB some component may be reused for developing App.
- Documented Architecture

Automating Infrastructure Deployment

key concepts

- Provisoning resources
- planning updates
- using source control
- Re using templates

First

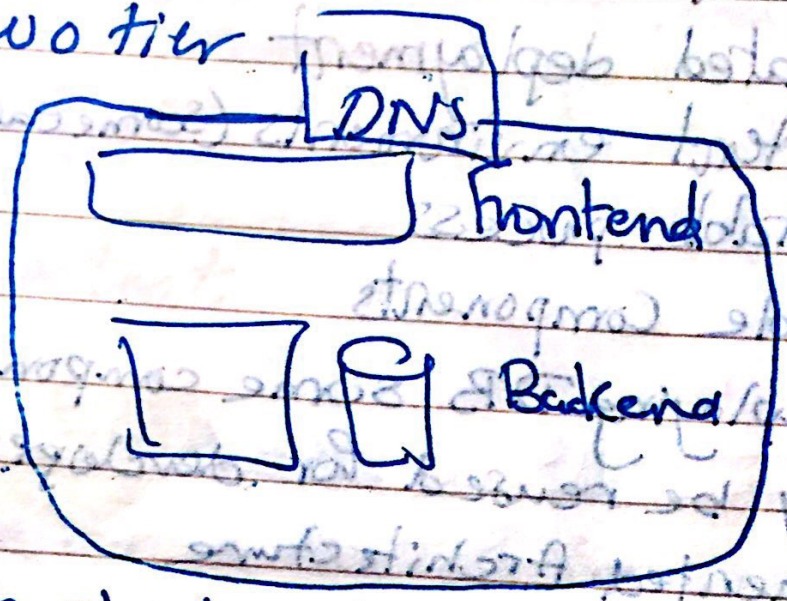
Provisioning Resources

Our Scenario:- You're asked to provision new development environment be part of new lic business app that Company is

Scanned with CamScanner

into a SaaS product

Two tier



using cloud environment to deploy apps so you'll use AWS, you can setup it manually, but someone told you about new s/w called Terraform

Terraform Components

Terraform Exe	Terraform files	Terraform plugins	Terraform state
	Configs .tf	interact with providers like AWS	what's going on "state"

It's not a good practice to store your AWS Credentials into terraform files, so Terraform gives you variables to define them

5

Variables

```
variable "aws-access-key" {}
variable "aws-secret-key" {}
variable "aws-region" {
    default = "us-east-1"
}

provider "aws" {
access_key = "var.access_key"
secret_key = "var.secret_key"
region = "var.aws-region"
}
```

Data

```
data "aws_ami" "alx" {
most_recent = true
owners = ["amazon"]
filters {}
```

Scanned with CamScanner

server to host

6

```
resource "aws_instance" "ex" {
  ami = "data.aws_ami.ubuntu.id"
  instance_type = "t2.micro"
```

Resource
variable
hardcode

```
output "aws_publicip" {
  value =
  "aws_instance.ex.public_dns"
}
```

output

Config files uses HashiCorp config files

.tf files

Example files in D:\ll courses

Amazon keys

First section : variables
Second : providers → like amazon
Third : Data
Fourth : Resources

virtual private cloud

resources

VPC_id = aws_default_vpc.default.id

defined in

Provider (m3/module-three.tf) (7)

defining aws provider to & give it access variables

Data ~~getting~~ calling data from provider

AMI ID Amazon Machine Image ID ⇒ includes OS & other sw and config info to run Instance

EC2 Amazon Elastic Compute Cloud

Resource

1. AWS vpc Resource to use default vpc in region

2. Aws security group to connect via SSH to our instance

- Two ingress rules to Allow X ports traffic to 80, 22
- on egress? to allow traffic from instance to internet

Scanned with CamScanner

3- aws_instance nginx

provisioner uses aws_instance to remotely execute something

output
what Terraform gives us!

Get aws plugin
terraform init

terraform plan -out m3.tfplan
↳ get the diff between actual and new and load variables

terraform apply "m3.tfplan"

JSON ⇒ state of Terraform
only read it not modify
- Resource Mapping
- Locking ⇒ during changes
- Location ⇒ local (default), remote (aws, azure, ...)

- ~~Workspaces~~

(9)

- Workspace: one file for each (later)

{ "version": 4,
"Terraform version": "0.12.5",
"serial": 30
outputs: {},
resource: [],
}

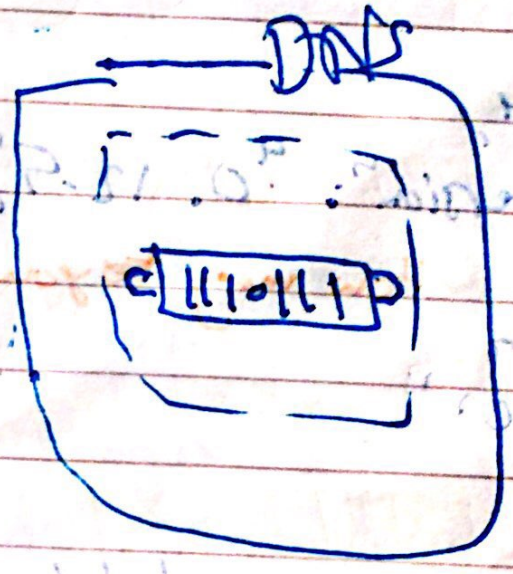
How many times you run plan/apply

Terraform planning (What happens inside)

- Inspect state
- Dependency graph
- Additions, updates and deletions
- parallel Execution
- save the plan

Scanned with CamScanner

provision new vpc to host App
Add vpc to sub net and give DNS



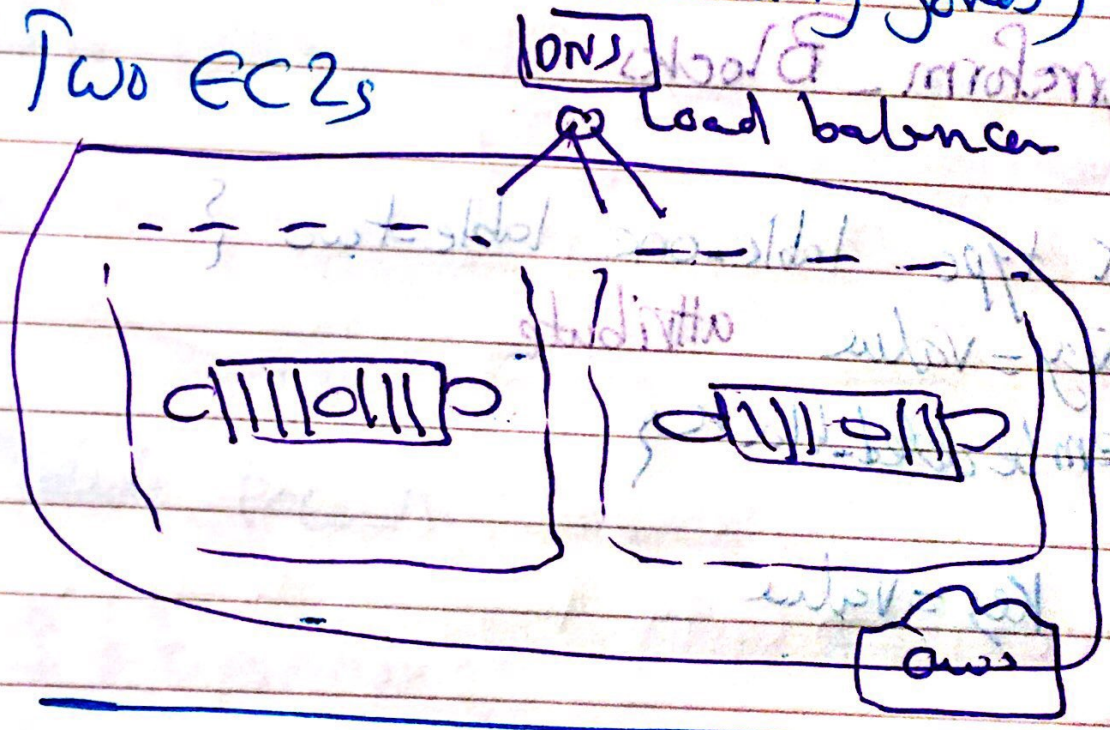
Adding vpc

```
resource "aws_vpc" "vpc" {}
resource "aws_internet_gateway" "igw" {}
resource "aws_subnet" "subnet1" {}
resource "aws_route_table" "rtb" {}
resource "aws_route_table_association" "rtb-subnet1"
{}
```

Deploying VPC ⇒ Notes available in'd file as comments on my gt hub

We need to support multiple instances (different availability zones)

Two EC2s



M5:. Configuring Resource after Creation using provisioner.

Scanned with CamScanner

Terraform ⇒ HashiCorp Config Language (12)

Why not JSON ⇒ not human readable & not editable

Not YML ⇒ to add more functionalities

Terraform_Blocks

```
block_type lable-one lable-two {
  key=value attribute
  embedded-block {

    key=value

  }
}
```

Object Types

string = "taco"

number = 5.5

bool = true

list = ["bean-taco", "beef-taco"]
↑0 ↑1

Scanned with CamScanner

map = {name = "Ned", age = 42, love_taco = true}

Refrences

var. taco_day ⇒ Terraform knows it's variable

aws_instance. taco_truck. name

local. taco_topping. cheeses ⇒ locally defined variables in the config.

Map of key-choices

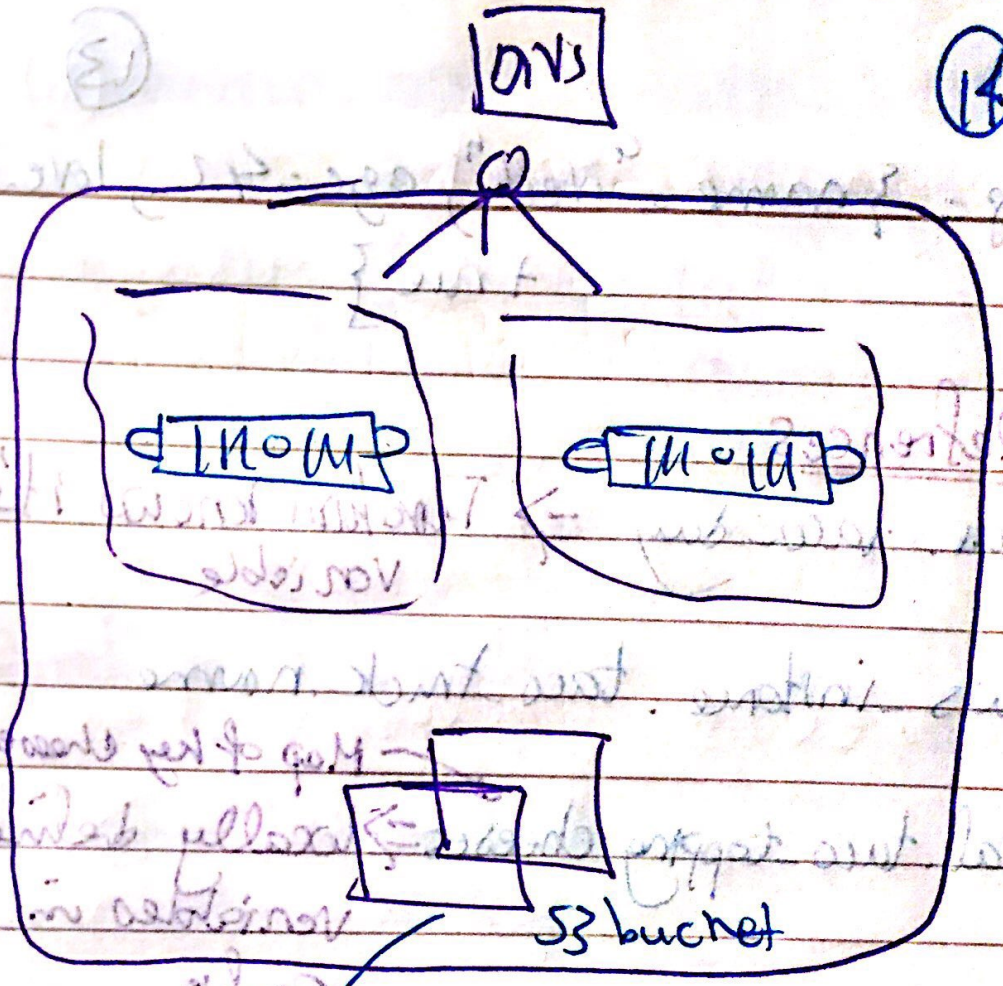
module. taco_hut. locations

interpolation

taco_name = "Neds ${var.taco_type}"

Concatenate

Scanned with CamScanner



Contains Common HTML files & Logs ⇒ with tagging to know log source

Provisioners (creators) ⇒ Post deploy Config.
(last resort)
↳ fine and creation or destruction

Example from local machine to remote

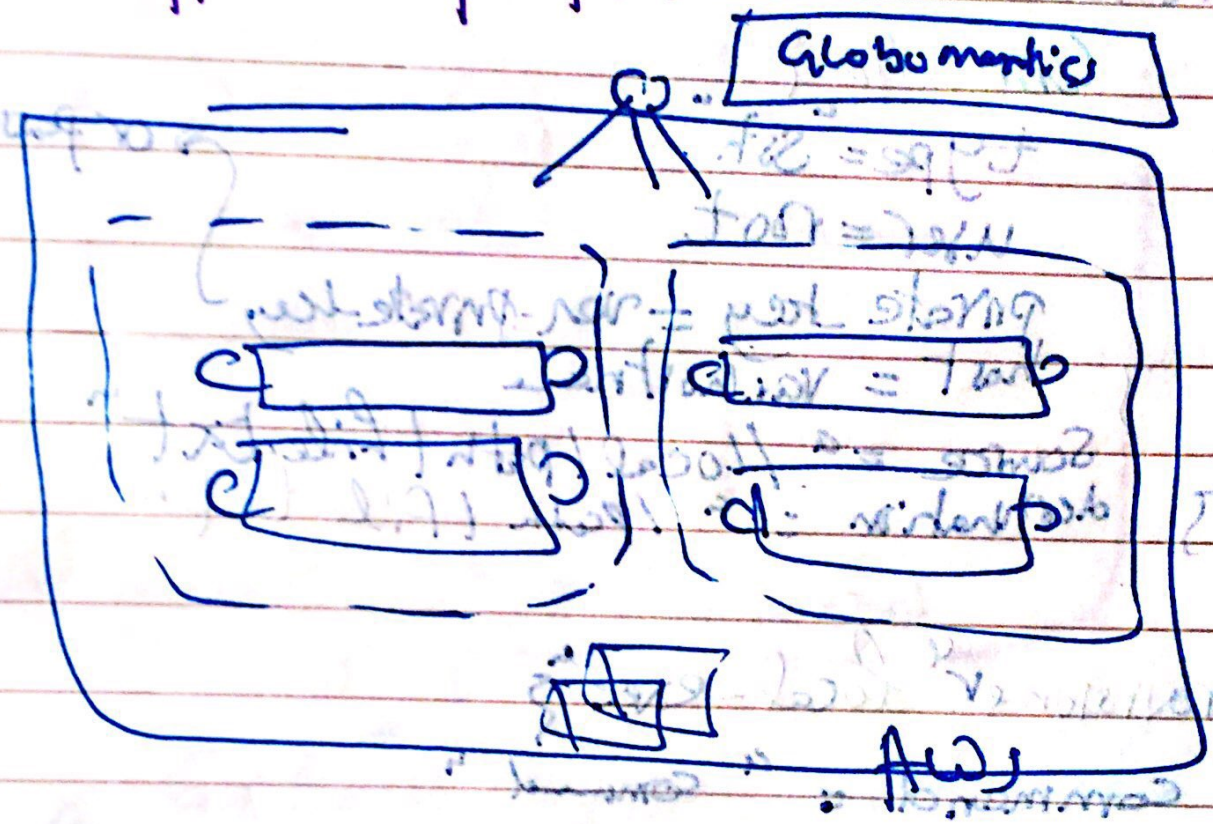
```
provisioner "file" {
  connection {
    type = "ssh"
    user = root
    private_key = var.private_key
    host = var.hostname
  }
  source = "/local/path/file.txt"
  destination = "/path/file.txt"
}

provisioner "local-exec" {
  command = "command"
}

provisioner "remote-exec" {
  scripts = ["list", "of", "local", "scripts"]
}
```

or password

Scanned with CamScanner

Adding New Provider

Support Multiple Providers



Req:

- Make the Number of instances of EC2 dynamic
- Replace the load balancer with Globomantics.xyz hosted on Azure

Functions Built-in Terraform

funcName(arg1, arg2, ...)

Fn Main Categories

1- Numeric ⇒ Manipulate numbers

min(42, 13)

2- string ⇒ lower("TAco")

3- Collection → lists

→ Maps

merge(map1, map2)

4- Filesystem ⇒

file(path) Read contents to string

5- IP Networking

cidrsubnet()

6- Date & Time ⇒ timestamp()

Scanned with CamScanner

```
variable network_info {
  default = "10.1.0.0/16"
}

# Returns 10.1.0.0/24

cidr_block = cidrsubnet(
    var.network_info, 8, 0)
```

Add 8 to Mask

First address we need

```
# Returns 10.1.0.5

host_ip = cidrhost(var.network_info, 5)
```

Maps

map name

```
ami = lookup(var.ami, "us-east-1",
    "error")
```

default value if not found in

value to find

Terraform providers

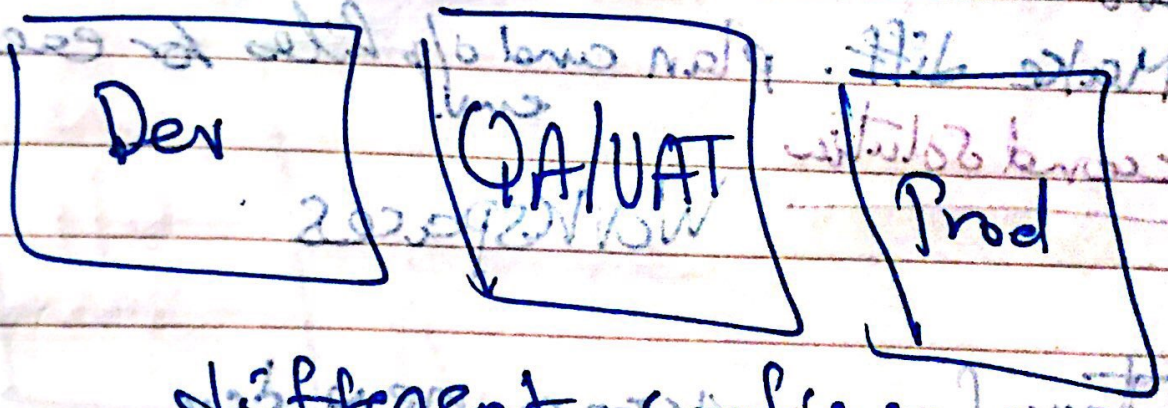
IaaS PaaS SaaS

Can be written by ⇒ (Many)

Community HashiCorp

Info Can be get from providers github pages

Deploying in additional environments



different configs.

First Solution: Working with variables

Sources:
- file
- environment variables
- var option command line

Environment variable
File
Command line ⇓ highest priority

You will Variable ~~of~~ files for each environment

In the github

one Config file split into

Variables.tf resources.tf output.tf

Make diff. plan and o/p files for each env.

Second solution

Workspaces

Terraform workspace new dev

Storage of secrets:

Variable files, Environment Variable, Through Command line OR Secret Management soln. AZure, Amazon

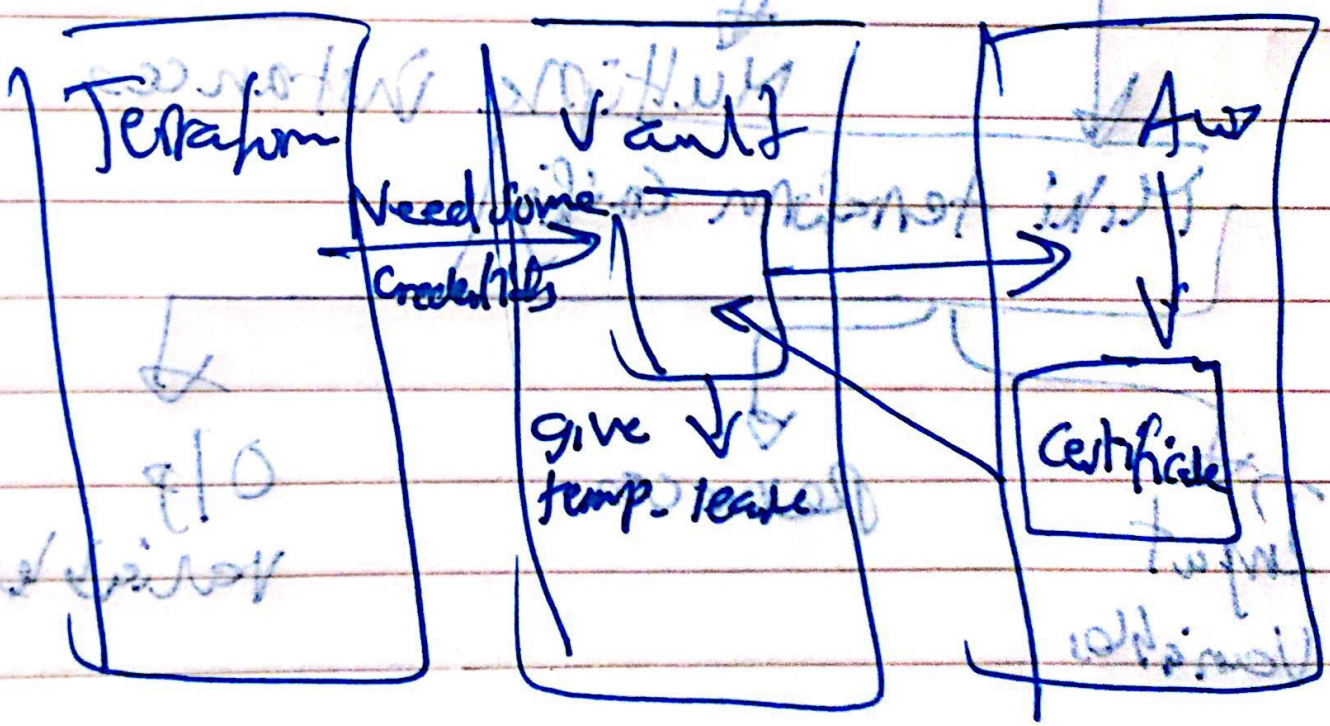
Example Environment Variable

21

Powershell

$env:AWS_Access_key_Id=
"ASSIXY30M"

Vault (HashiCorp)



Module ⇒ Can be reused in diff. apps

Scanned with CamScanner

Module

- Code reuse
- Remote or Locally

↓

Terraform registry

↓

Modules support versioning

↓

provider inheritence

↓

Multiple instances

↓

Mini terraform config

- Input Variables
- Resources
- O/p variables