Terraform

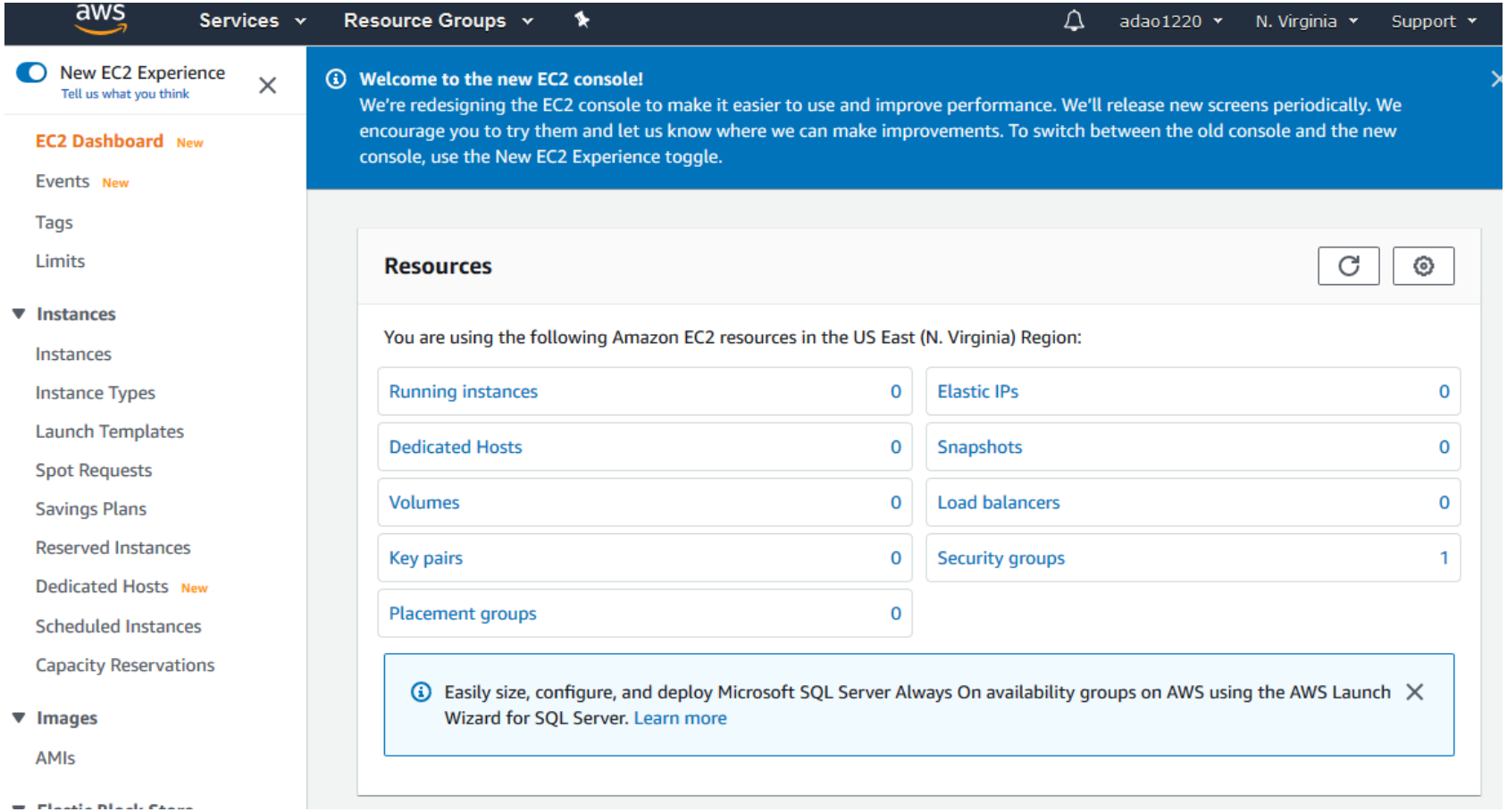
```
provider "aws" {
  region     = var.aws_region
  access_key = "Welcome to the presentation!!"
  secret_key = "I hope you can learn something from this"
  version    = ">= 2.6.0"
}
```

ROCKET Technology

Overview

- Infrastructure of Code
- Why Terraform?
- Optimizing with-in Terraform
- How can it help you (Why does this matter to you?)
- Cons and Pros

Infrastructure of Code (IaC)



Iter by: All instance types | Current generation | Show/Hide Columns

Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-O
☐	General purpose	t2.nano	1	0.5	EBS only	
☑	General purpose	t2.micro Free tier eligible	1	1	EBS only	
☐	General purpose	t2.small	1	2	EBS only	
☐	General purpose	t2.medium	2	4	EBS only	
☐	General purpose	t2.large	2	8	EBS only	
☐	General purpose	t2.xlarge	4	16	EBS only	
☐	General purpose	t2.2xlarge	8	32	EBS only	
☐	General purpose	t3a.nano	2	0.5	EBS only	

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security from an existing one below. Learn more about Amazon EC2 security groups.

Assign a security group: ◉ Create a **new** security group

○ Select an **existing** security group

Security group name: launch-wizard-1

Description: launch-wizard-1 created 2020-08-04T12:53:23.781-04:00

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	Custom 0.0.0.0/0	e.g. SSH for Admin Des

Add Rule

Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to ta management role to the instance, and more.

Number of instances 1 Launch into Auto Scaling Group

Purchasing option ☐ Request Spot instances

Network vpc-49fa1434 (default) Create new VPC

Subnet No preference (default subnet in any Availability Zone) Create new subne

Auto-assign Public IP Use subnet setting (Enable)

Placement group ☐ Add instance to placement group

Capacity Reservation Open

IAM role None Create new IAM ro

Shutdown behavior Stop

Stop - Hibernate behavior ☐ Enable hibernation as an additional stop behavior

Enable termination protection ☐ Protect against accidental termination

Monitoring ☐ Enable CloudWatch detailed monitoring
Additional charges apply.

Tenancy Shared - Run a shared hardware instance
Additional charges may apply when launching Dedicated instances.

Elastic Inference ☐ Add an Elastic Inference accelerator
Additional charges apply.

T2/T3 Unlimited ☐ Enable
Additional charges may apply

File systems Add file system Create new file system

▼ Advanced Details

Metadata accessible Enabled

(ALL manually done) 1-2 weeks

Infrastructure of Code (IaC)

```
provider "aws" {
  region              = var.aws_region
  allowed_account_ids = [var.aws_account_id]
}

data "aws_caller_identity" "current" {}

data "aws_kms_key" "default_secretsmanager_key" {
  count  = var.use_kraken_secret_store ? 1 : 0
  key_id = "alias/aws/secretsmanager"
}

data "aws_kms_key" "default_ssm_key" {
  count  = var.use_kraken_parameter_store ? 1 : 0
  key_id = "alias/aws/ssm"
}

data "aws_kms_alias" "kms_key_alias" {
```

Benefits of using IaC:

1. Version control
2. Test
3. Review
4. Reuse
5. Automate

Three IaC Tools :

Cloud Development Kit (Python, Java)
CloudFormation (YAML, JSON)
Terraform (HCL)

HCL – Hashicorp Configuration Language

- “Not really code”… but a configuration language
  - Human readable as well as machine friendly
- Contains resources(services) and configurations
- Doesn’t use tab system, Uses two spaces
- Variables you can use:
  - String
  - Number
  - List
  - Map
  - …

Why Terraform?

Terraform

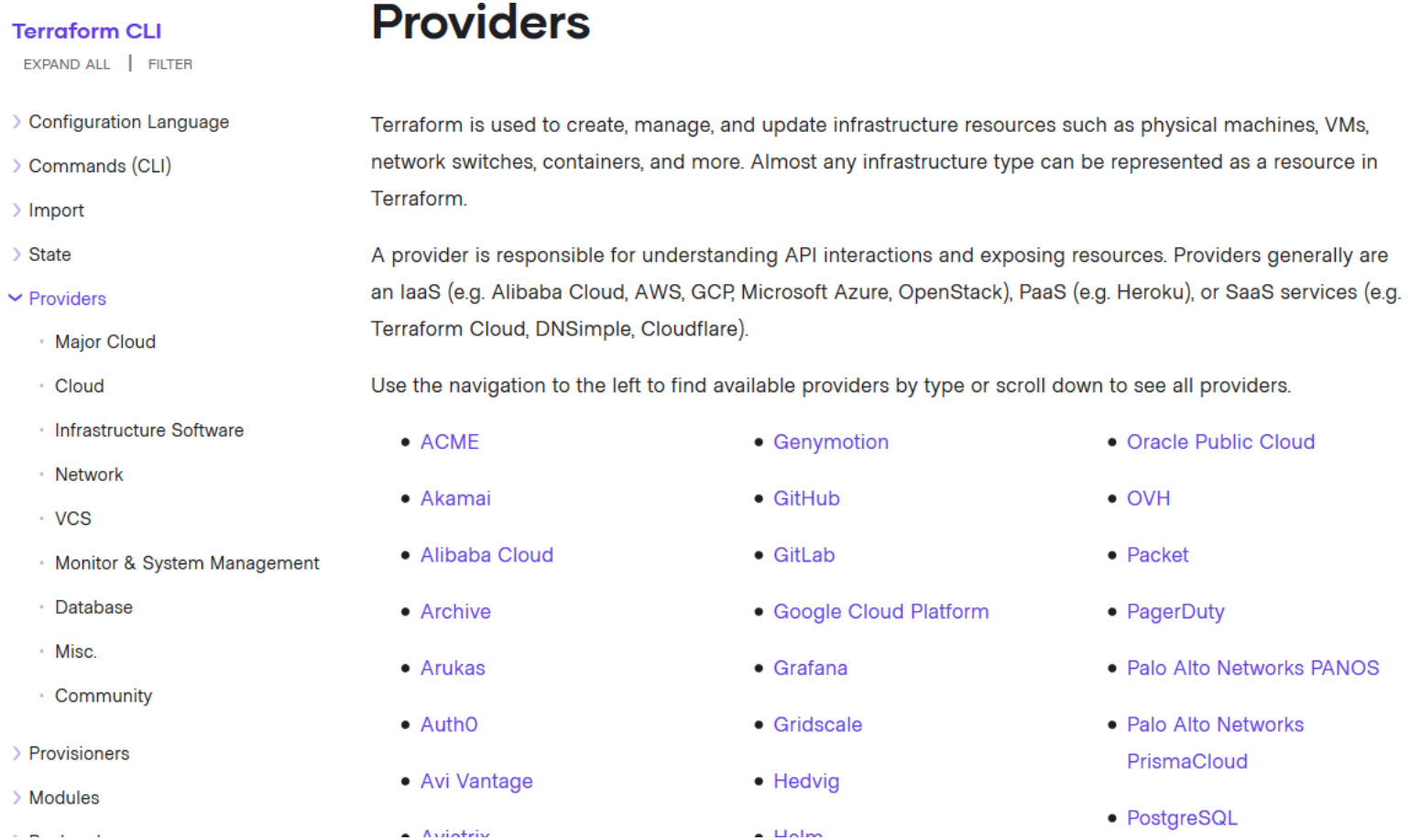
```
resource "aws_instance" "ec2"{
    ami = "ami-0e90897638287 57e1"
    instance_type = "t2.micro"
}

resource "aws_eip" "EleasticEIP"{
    instance = aws_instance.ec2.id
    #attaches to the EC2
}
```

-Similar to JSON (CloudFormation)

-Human-readable

-Supports Many Providers

Why Terraform?



Why Terraform?

Terraform

```
resource "aws_instance" "ec2"{
    ami = "ami-0e90897638287 57e1"
    instance_type = "t2.micro"
}

resource "aws_eip" "EleasticEIP"{
    instance = aws_instance.ec2.id
    #attaches to the EC2
}
```

-Similar to JSON (CloudFormation)

-Human-readable

-Supports Many Providers

-Modules!

Short little demo on how we can use Terraform

To save time, I already written the code to do an EC2, with a ElasticIP, and outputting the Elastic Public IP

Desired State vs Current State

Desired State: The configurations with Terraform

```
#Creating the EC2 instance
resource "aws_instance" "ec2" {
  ami           = "ami-0e9089763828757e1"
  instance_type = "t2.micro"
  user_data     = file("server-script.sh")
  tags = {
    name = "PresentationTime"
  }
```

Desired State

Current State: What is currently posted on to the console

-Run “terraform refresh”

-Check terraform.tfstate

Terraform will always want to set it to the DESIRED STATE

Desired State vs Current State DEMO

In Fun Code Terms

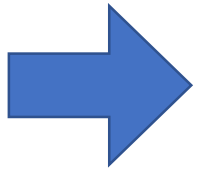
```
If (Desired State != Current state){
        resets (*)everything to the desired state
}
```

(*) well… Almost everything

Optimizing with-in Terraform

main.tf(original)

```
provider "aws" {
  region     = "us-east-1"
  access_key = "AKIAVVWBGK63DRX4L7XF"
  secret_key = "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"
  version    = ">= 2.6.0"

}

#Creating the EC2 instance
resource "aws_instance" "ec2" {
  ami           = "ami-0e9089763828757e1"
  instance_type = "t2.micro"
  tags = {
    name = "PresentationTime"
  }
}
```



```
provider "aws" {
  region     = var.aws_region
  access_key = var.access_key
  secret_key = var.secret_key
  version    = ">= 2.6.0"

}

#Creating the EC2 instance
resource "aws_instance" "ec2" {
  ami           = var.ami
  instance_type = var.Instance_type
  tags = {
    name = var.tag_name
  }
}
```

Optimizing with-in Terraform

main.tf

```
provider "aws" {
  region     = var.aws_region
  access_key = var.access_key
  secret_key = var.secret_key
  version    = ">= 2.6.0"

}

#Creating the EC2 instance
resource "aws_instance" "ec2" {
  ami           = var.ami
  instance_type = var.Instance_type
  tags = {
    name = var.tag_name
  }
}
```

variables.tf

```
#-----------------------------------------------------------
#      VARIABLES FOR THE PROVIDER
#-----------------------------------------------------------

variable "aws_region" {
  type        = string
  description = "The region that we are in on AWS"
  default = "us-east-1"
}
variable "access_key" {
  type        = string
  description = "the access key to the AWS"
  default = "AKIAVVWBGK63DRX4L7XF"
}
variable "secret_key" {
  type        = string
  description = "the access key to the AWS"
  default = "sQi6EARvv4mlvQO21f32gmyPXRfn3qeKpMGMmn3e"
}
```

terraform.tfvars

```
aws_region    = "us-east-1"
access_key    = "AKIAVVWBGK63DRX4L7X
secret_key    = "sQi6EARvv4mlvQO21f3
ami           = "ami-0e9089763828757
Instance_type = "t2.micro"
EC1Name       = "Using value from Ter
```

Optimizing with-in Terraform (More…)

Modules

- One of the core features of Terraform
- Encapsulates a piece of configuration
    - Module takes-in input variables
    - Can return output variables

```
module "security" {
  source       = "../Security"
  SecurityName = "Allowing HTTP"
  ingress      = var.ingress
  egress       = var.egress
}
```

Security module code

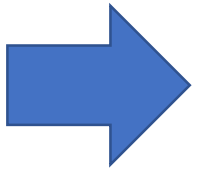
```
resource "aws_security_group" "webtraf" {
  name = var.SecurityName
  dynamic "ingress" {
    iterator = port
    for_each = var.ingress
    content {
      from_port   = port.value
      to_port     = port.value
      protocol    = "TCP"
      cidr_blocks = ["0.0.0.0/0"]
    }
  }
  dynamic "egress" {
    iterator = port
    for_each = var.egress
    content {
      from_port   = port.value
      to_port     = port.value
      protocol    = "TCP"
      cidr_blocks = ["0.0.0.0/0"]
    }
  }
}

output "sgName" {
  value = aws_security_group.webtraf.name
}
```

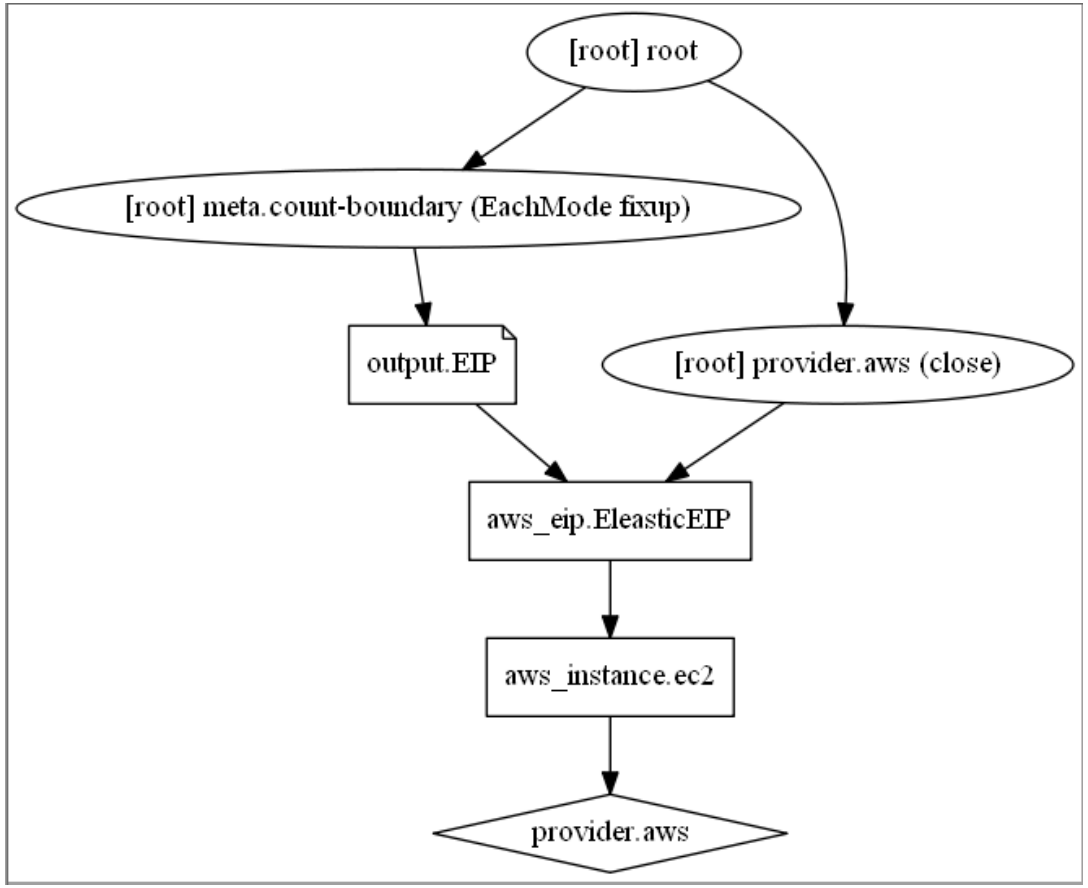
Mutli-Cloud Management with Terraform

```
terraform {
  required_providers {
    aws = "~> 2.39.0"
    google      = "~> 2.18.0"
    azurerm = "~> 1.41.0"
  }
}

provider "aws" {
  region = var.aws_region
}

provider "google" {
  credentials = file(pathexpand("~/.config/gcloud/${var.google_project_id}.json"))
  region      = var.google_region
  project     = var.google_project_id
}

provider "azurerm" {
  azure_region = "eu-west"
}
```

Side Fun Stuff (Terraform Graph)

```
provider "aws" {
    region      = "us-east-1"
    access_key = "[redacted]"
    secret_key = "[redacted]"
}

resource "aws_instance" "ec2"{
    ami = "ami-0e9089763828757e1"
    instance_type = "t2.micro"
}

resource "aws_eip" "EleasticEIP"{
    instance = aws_instance.ec2.id
    #attaches to the EC2
}

output "EIP"{
    value = aws_eip.EleasticEIP.public_ip
    #getting the public IP
}
```





http://www.webgraphviz.com/

How can it help you (Why does this matter to you?)

- Mutli-Cloud Management
- Learning one provider... You learnt them all

AWS

```
module "web_server_sg" {
  source = "terraform-aws-modules/security-group/aws//modules/http-80"

  name        = "web-server"
  description = "Security group for web-server with HTTP ports open within VPC"
  vpc_id      = "vpc-12345678"

  ingress_cidr_blocks = ["10.10.0.0/16"]
}
```

https://registry.terraform.io/modules/terraform-aws-modules/security-group/aws/3.13.0

Azure

```
module "network" {
  source              = "Azure/network/azurerm"
  resource_group_name = azurerm_resource_group.test.name
  address_space       = "10.0.0.0/16"
  subnet_prefixes     = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  subnet_names        = ["subnet1", "subnet2", "subnet3"]

  tags = {
    environment = "dev"
    costcenter  = "it"
  }
}
```

https://registry.terraform.io/modules/Azure/network/azurerm/3.1.1

Cons and Pros

Cons

1. Error handling and roll back
2. Desired state VS current state
    - Terraform + Manual Change = DON'T DO IT
3. Major updates → may break your code
    - Stick to a version

Pros

1. Readable
2. Desired state VS current state
3. Fairly easy to learn (not really code)
4. Learn one provider, learn them all
5. Cross-cloud provider capability
6. Terraform + Hal9000

Take away (Last words)

Terraform. Useful AF
Multicloud management
Can automated your services

Resources

- https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance
- https://medium.com/@pavloosadchyi/terraform-patterns-and-tricks-i-use-every-day-117861531173
- https://globaldatanet.com/blog/cloudformation-vs-terraform
- https://www.hashicorp.com/resources/what-is-infrastructure-as-code/

- Mutli-Cloud VPN with Terraform
  - https://github.com/Silectis/multi-cloud-vpn

Thank you