OpenCredo
A TRIFORK COMPANY

# Misadventures With Terraform

**Matthew Revell**
Senior DevOps Consultant

@nightowlmatt

#CodeMeshLDN

# Introduction To Terraform

In 5 minutes or less

#CodeMeshLDN

# What is Terraform?

Terraform is an Infrastructure as Code product from Hashicorp.

Used to automate provisioning of cloud infrastructure, SaaS, and other software.

Uses a plugin framework, called ‘providers’ to support a wide range of vendors.

# What is Terraform?

Terraform is an Infrastructure as Code product from Hashicorp.

Used to automate provisioning of cloud infrastructure, SaaS, and other software.

Uses a plugin framework, called ‘providers’ to support a wide range of vendors.

# Terraform Resources

```
provider "aws" {
  version = "~> 2.26"
}

resource "aws_vpc" "example" {
  cidr_block           = "10.0.0.0/16"
}
```

# Terraform Modules

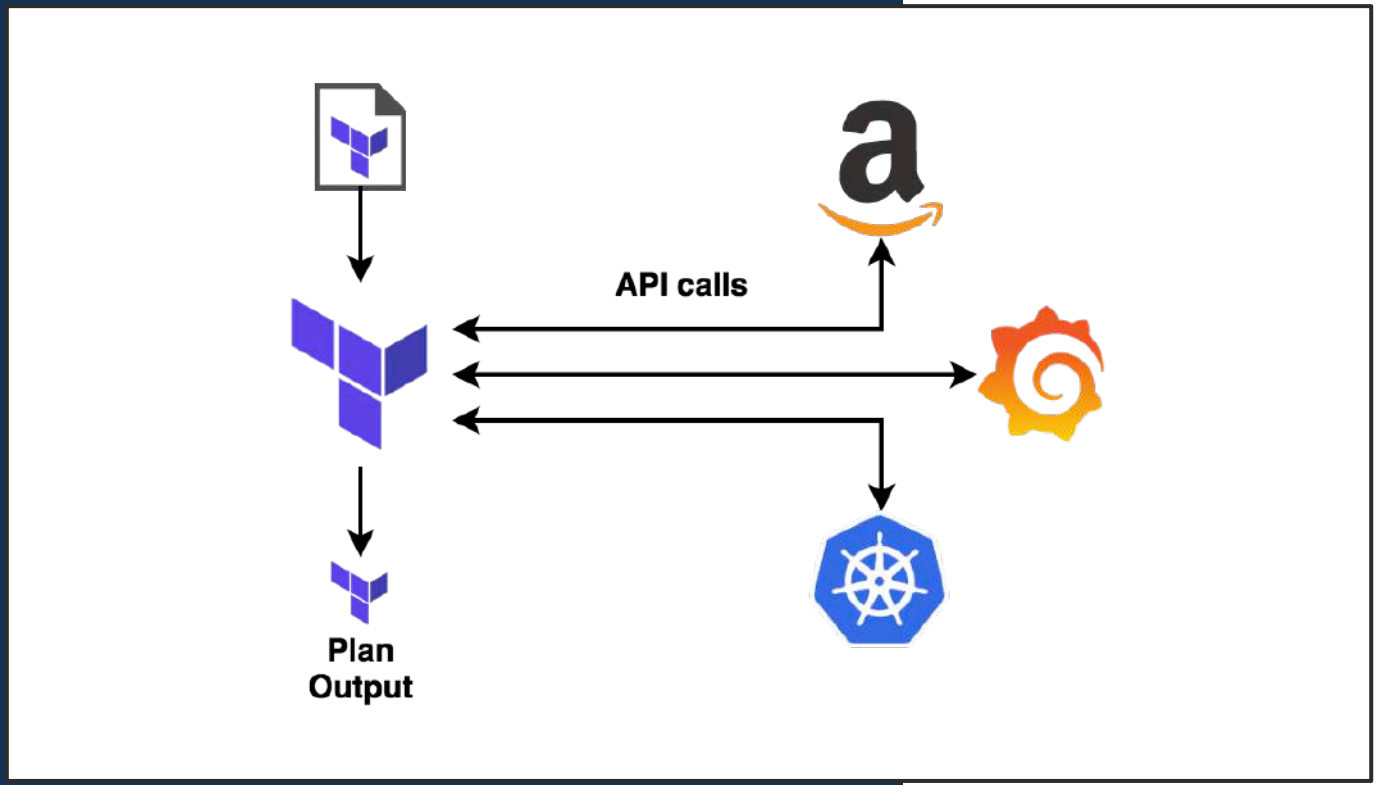
```
module "local" {
  source        = "../modules/app"
  instance_type = "t3.medium"
}

module "git" {
  source        = "git::https://example.com/app.git?ref=0.4.20"
  instance_type = "m4.xlarge"
}
```

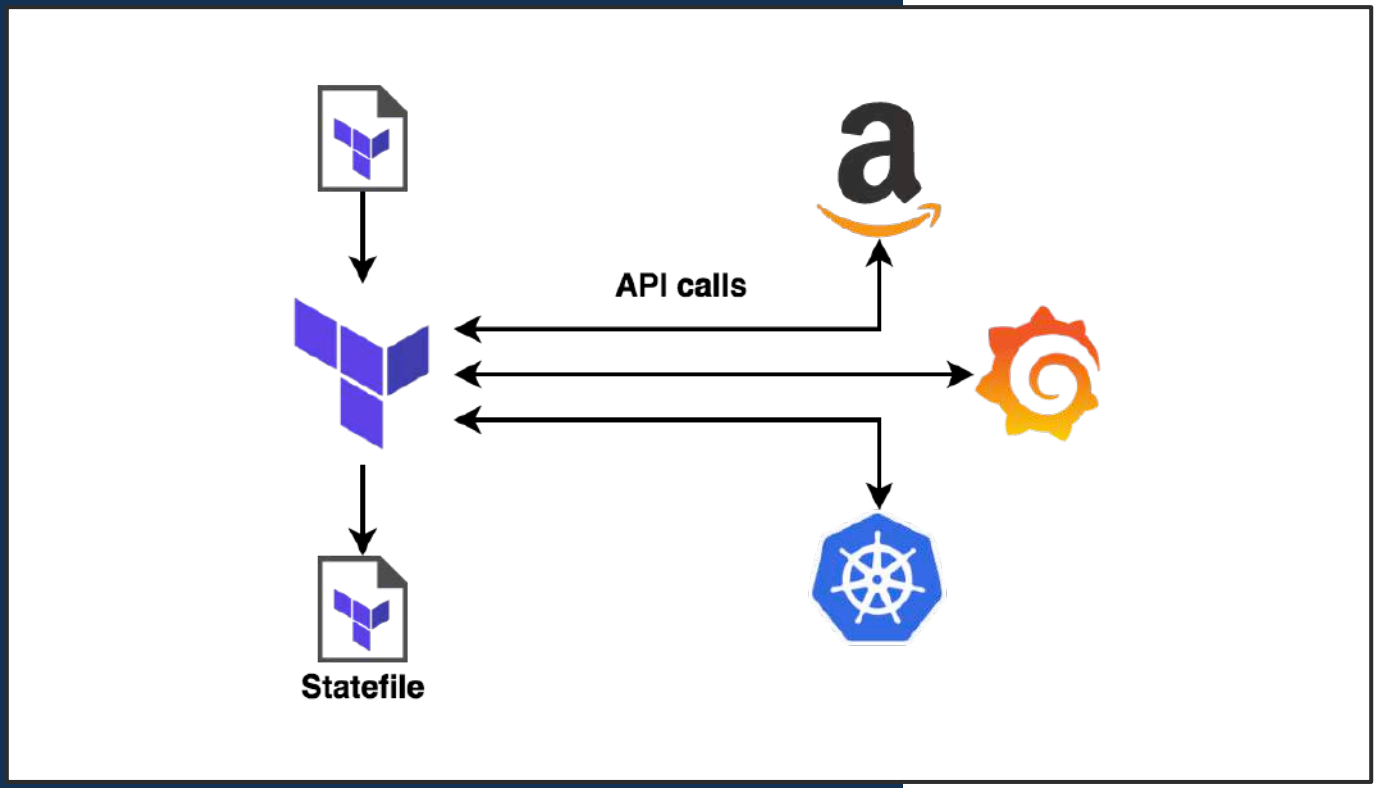
# Terraform Statefile

```
terraform {
  backend "s3" {
    bucket = "terraform-states"
    key    = "example/terraform.tfstate"
  }
}
```

# Terraform Plan



# Terraform Apply

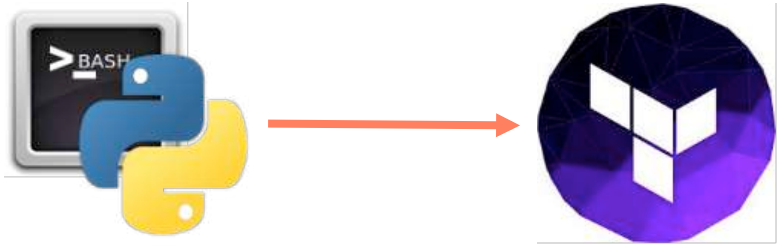


# How Did I Get Here?

# In the beginning...


**Bash &**
**Python**

# A few years ago...




**Bash & Python**

**Terraform 0.6.xx**

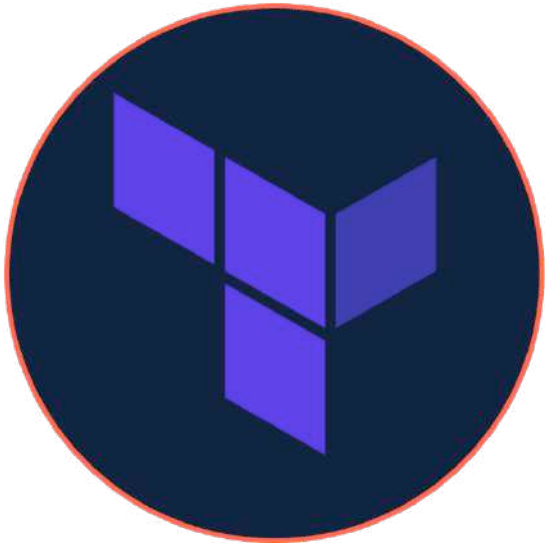
# Into the future...

# Honey, I Shrunk The Modules

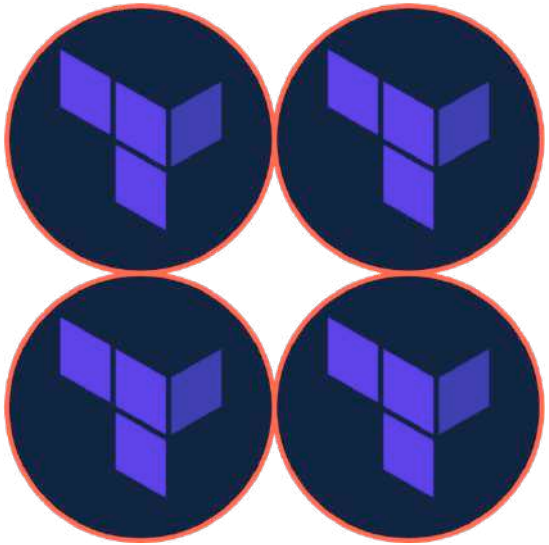
when modules get too small

#CodeMeshLDN

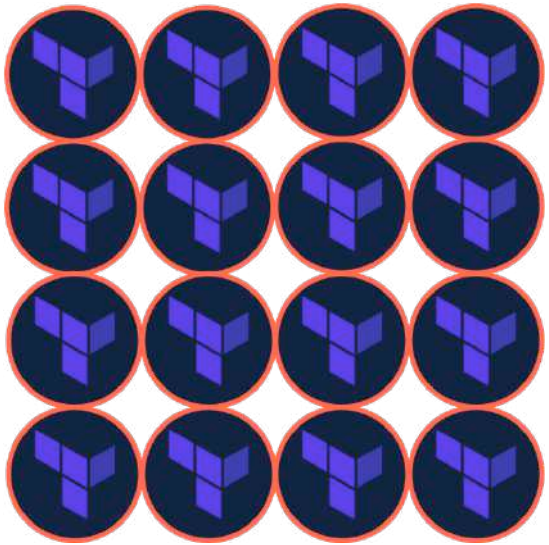
# Monolith Terraform State



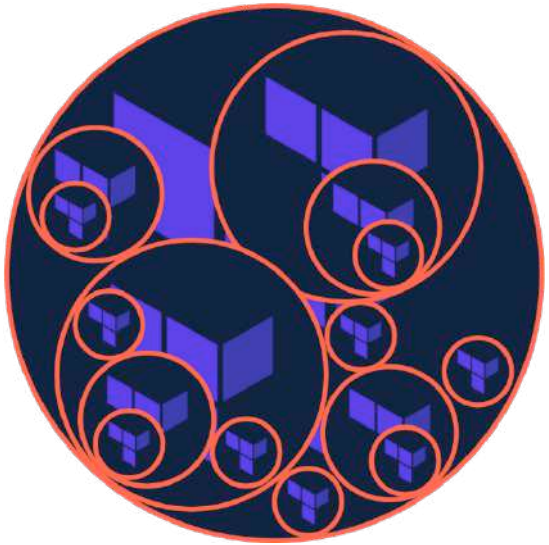
# Reusable Modules



# Single Resource Modules



# Complex Deployment



# Lessons Learned (small modules)

Consider whether a single resource module adds any value

**Value**

Consider whether the additional complexity is worth the perceived value

**Complexity**

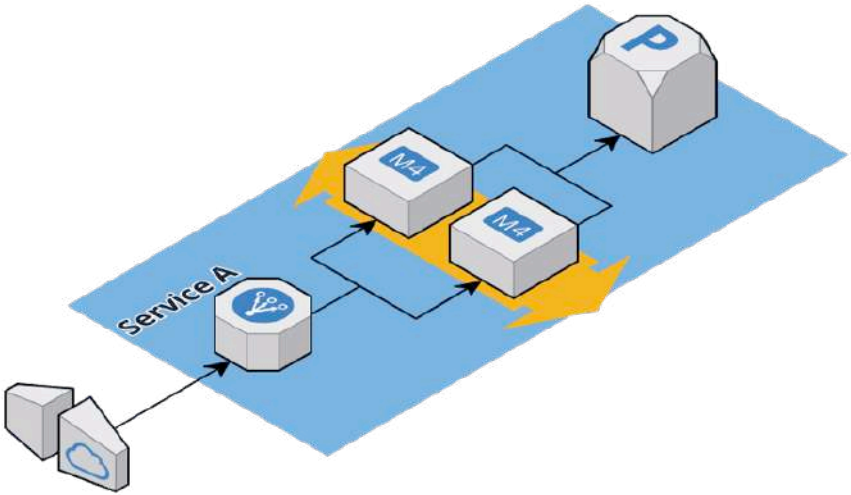
Consider whether the module will be usable by the intended consumer(s)

**Usability**

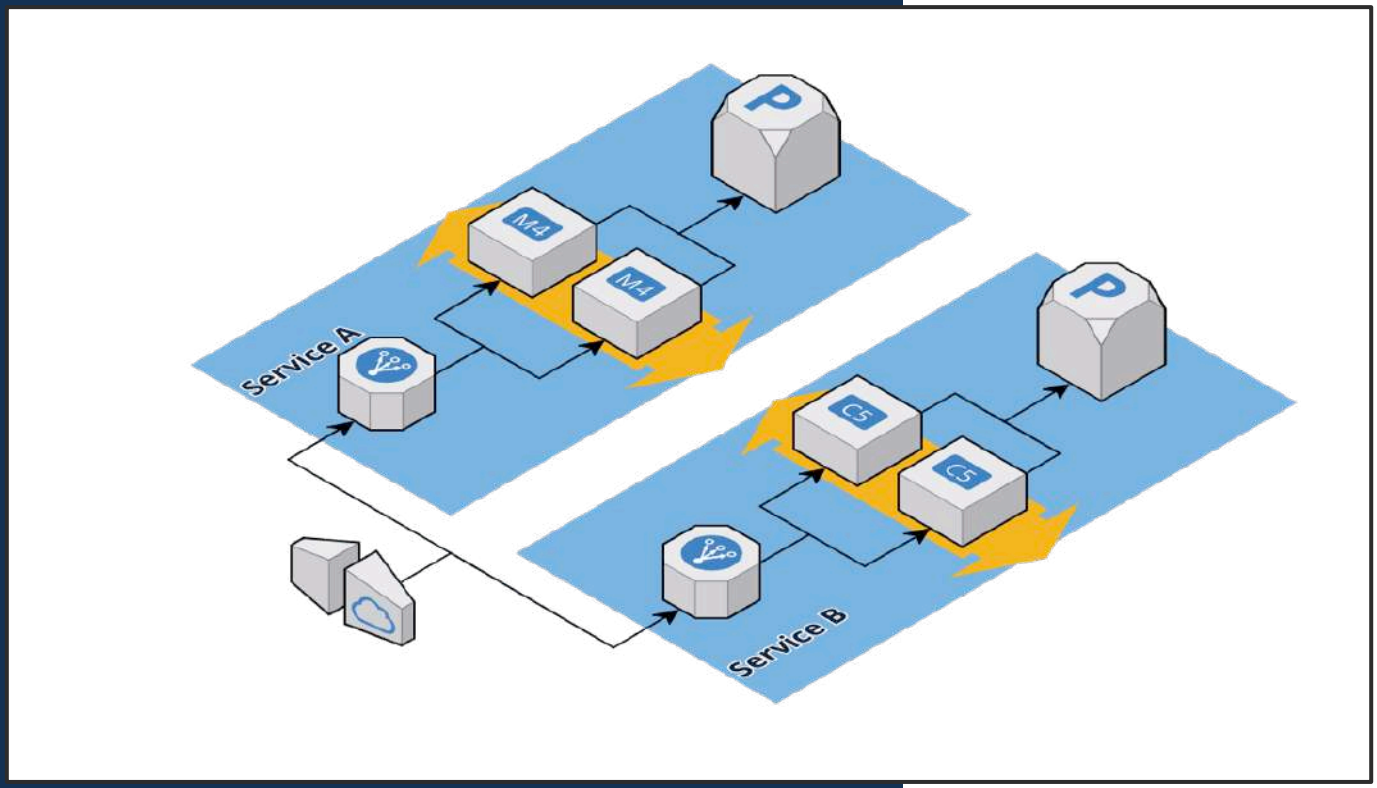
# Run Once And Forget

pray the code works the second time

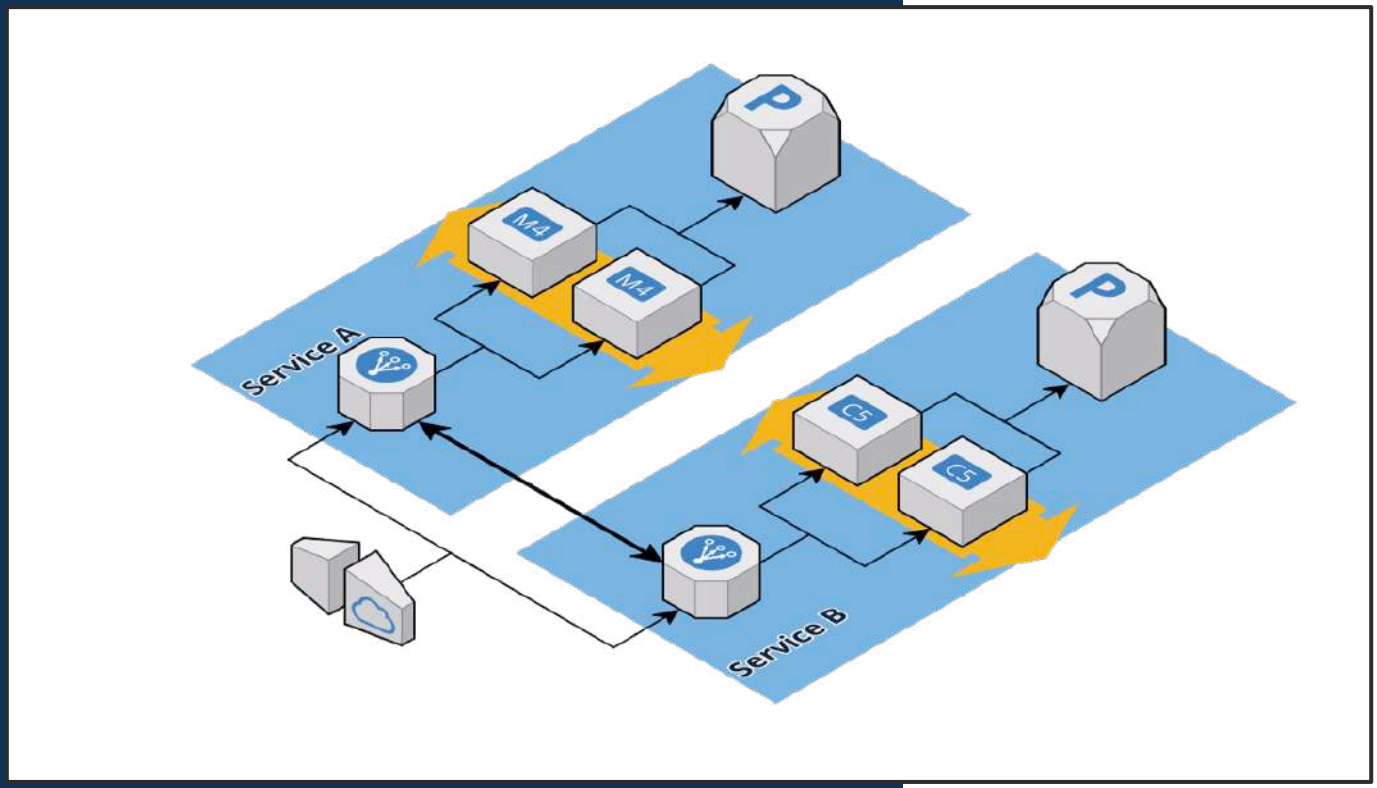
# A Service



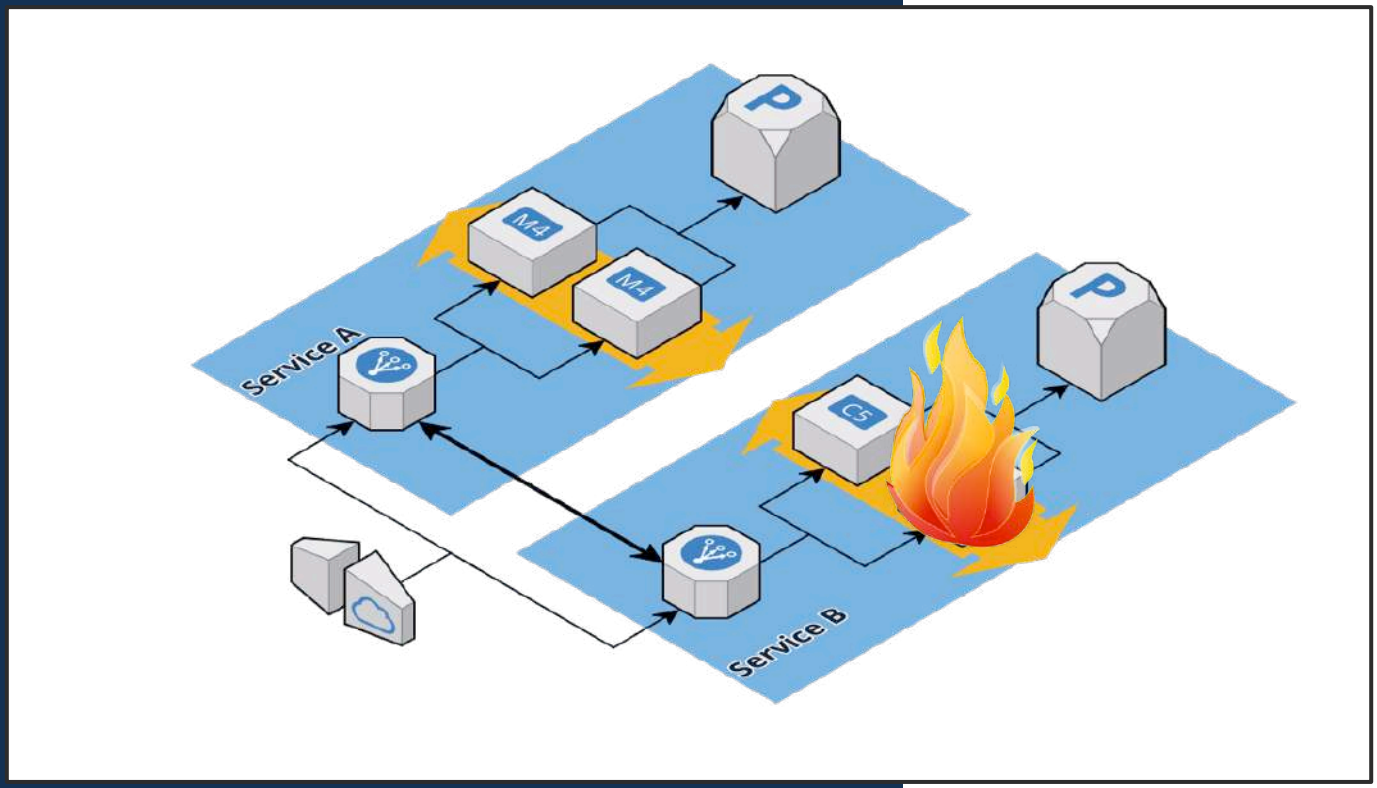
# Two Services



# Two Connected Services



# A Service On FIRE!



# Lessons Learned (running code once)

Testing modules in isolation can only validate the internals

**Testing**

Full deployment tests are essential to validate the entire Terraform structure

**Testing**

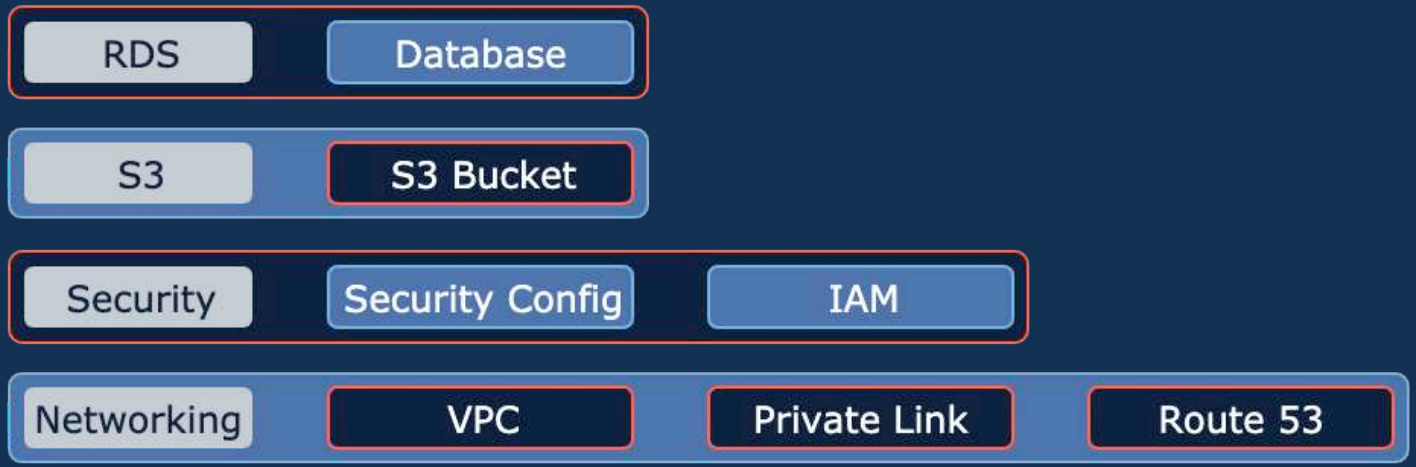
A dedicated account can allow continuous testing without disruption

**Testing**

# How To Slice The Cake

in this case dividing Terraform States

# Terraform States and Modules



# Terraform States and Modules

# Lessons Learned (dividing states)

| | | |
|---|---|---|
| Resources should be grouped such that states do not grow exponentially | States should have a limited scope to minimise impact in the event of mistakes | Teams should be able to manage their own Terraform independently |
| **Scalability** | **Blast Radius** | **Ownership** |

# To *dir* Is Human

but *DRY* is divine

#CodeMeshLDN

# Terraform Code Repo

```
terraform
  + env
    + dev
      - terraform.tfvars
      - backend.tf
      - main.tf
    + prod
      - terraform.tfvars
      - backend.tf
      - main.tf
```

# Terraform main.tf

```
variable "instance_type" {}

module "vpc" {
  source = "git::https://example.com/vpc.git?ref=0.1.0"
}
module "app" {
  source        = "git::https://example.com/app.git?ref=0.4.20"
  instance_type = var.instance_type
}
```

# Terraform terraform.tfvars

```
instance_type = "m4.large"

instance_count = "3"
```

# Terraform backend.tf

```
terraform {
  backend "s3" {
    bucket = "terraform-states"
    key    = "prod/terraform.tfstate"
  }
}
```

# Terraform Code Repo

```
terraform
  + env
    + dev
      - terraform.tfvars
      - backend.tf
      - main.tf
    + prod
      - terraform.tfvars
      - backend.tf
      - main.tf
```

# Terragrunt

- Thin wrapper for Terraform
- Allows for easier management of backends
- Reduces amount of repeated code
- Developed by Gruntworks

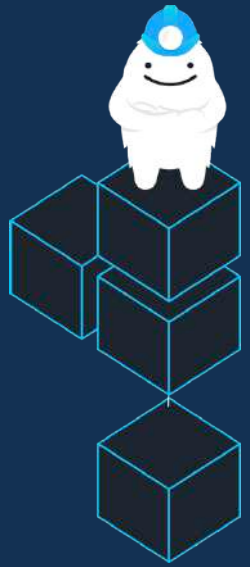


Image courtesy of Gruntworks Inc.

# Terragrunt Code Repo

```
terraform
  + env
    - common.tfvars
    + dev
      - terraform.tfvars
    + prod
      - terraform.tfvars
```

# Terragrunt common.tfvars

```
terragrunt = {
  remote_state {
    backend = "s3"
    config {
      bucket = "my-terraform-state"
      key    = "${path_relative_to_include()}/terraform.tfstate"
    }
  }
}

instance_type  = "m4.medium"
```

# Terragrunt terraform.tfvars

```
terragrunt = {
  include {
    path = "../common.tfvars"
  }

  terraform {
    source = "git::https://example.com/deployment.git?ref=v0.0.1"
  }
}

instance_type = "m4.xlarge"
```

# Terragrunt Code Repo

```
terraform
  + env
    - common.tfvars
    + dev
      - terraform.tfvars
    + prod
      - terraform.tfvars
```

# Lessons Learned (repo structures)

Repeated code and copy and pasting will definitely lead to mistakes

**Keep it DRY**

A lack of clarity and readability will also lead to confusion and mistakes

**Clarity**

Tooling can help maintain clean code in complex deployments

**Tooling**

# Additional tooling links

- **Terragrunt**
  **- Filling the gaps in Terraform**
  `https://github.com/gruntwork-io/terragrunt`

- **Atlantis**
  **- Bringing GitOps to Terraform workflows**
  `https://www.runatlantis.io`

- **Kapitan**
  **- General purpose templating engine**
  `https://kapitan.dev`

# Concluding Remarks

# Questions