

- Terraform Language Documentation

Courses:

- Terraform Course - Automate your AWS cloud infrastructure

Complete Terraform Course - From BEGINNER to PRO! (Learn Infrastructure as Code)

- `terraform apply --var-file=file_name`
- `terraform apply --var="db_user=myuser" --var="db_pass=secretpassword"`
- `$ terraform workspace list`
- `$ terraform workspace new prodaction`

Course Overview

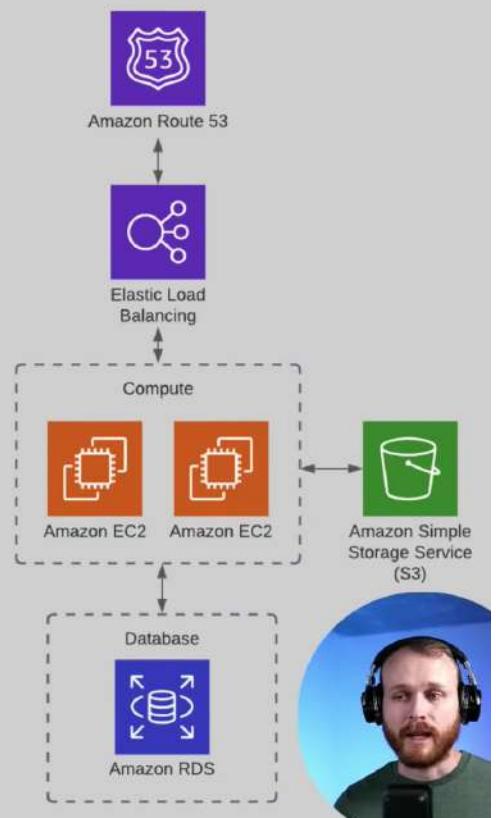
Timestamps in Description!

1. Evolution of Cloud + Infrastructure as Code
2. Terraform Overview + Setup
3. Basic Terraform Usage
4. Variables and Outputs
5. Language Features
6. Project Organization + Modules
7. Managing Multiple Environments
8. Testing Terraform Code
9. Developer Workflows



Reference Architecture

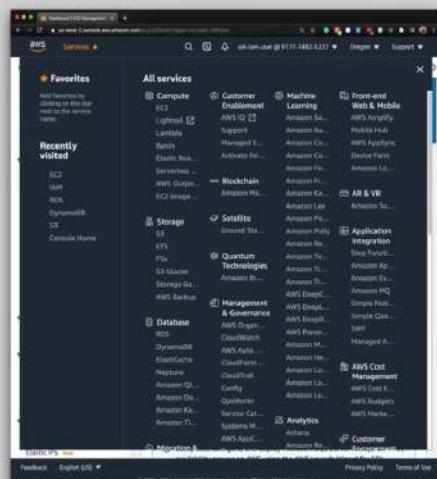
- Basic web application
- Infrastructure all within AWS
- Multiple instances running on EC2
- Using default VPC for simplicity



Screenshot of a GitHub repository page for 'sidpalas / devops-directive-terraform-course'. The repository is public. The main tab is selected, showing 4 branches and 5 tags. The code listing shows several commits from a user named 'sidpalas' [feature] Add workflow step to run terratest test (#5). The commits are dated July 5, 2021, and include updates to .github/workflows, 01-cloud-and-iac, 02-overview, 03-basics, 04-variables-and-outputs, 05-language-features, 06-organization-and-modules, 07-managing-multiple-environments, 08-testing, 09-developer-workflows, .gitignore, and README.md. The repository has 19 commits in total. On the right side, there is an 'About' section with a note 'No description, website, or topics provided.', a 'Readme' link, a '3 stars' rating, 1 watching, and 0 forks. There is also a 'Releases' section with 5 releases, the latest being 'Using event_name conditional' (Latest on May 27, 2021), and a 'Packages' section indicating no packages published.

Provisioning Cloud Resources

Three Approaches



GUI

```
Groups: []
Instances: [
  {
    "AmilaunchIndex": 0,
    "ImageId": "ami-011899242bb902164",
    "InstanceId": "i-0c451104d103db5e",
    "InstanceType": "t2.micro",
    "LaunchTime": "2021-01-11T15:51:00+00:00",
    "Monitoring": {
      "State": "disabled"
    },
    "Placement": {
      "AvailabilityZone": "us-east-1a",
      "GroupName": "",
      "Tenancy": "default"
    },
    "PrivateDnsName": "ip-172-31-81-227.ec2.internal",
    "PrivateIpAddress": "172.31.81.227",
    "ProductCodes": [],
    "PublicDnsName": "",
    "State": {
      "Code": 0,
      "Name": "pending"
    },
    "StateTransitionReason": "",
    "SubnetId": "subnet-cf8d2cc1",
    "VpcId": "vpc-15917f6f",
    "Architecture": "x86_64",
  }
]
```

API/CLI

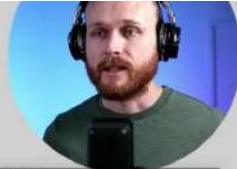
```
terraform {
  backend "s3" {
    bucket = "devops-directive-tf-state"
    key    = "Hello-World/terraform.tfstate"
    region = "us-east-1"
    dynamodb_table = "terraform-state-locking"
    encrypt     = true
  }

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = ">= 3.0"
    }
  }

  provider "aws" {
    region = "us-east-1"
  }
}

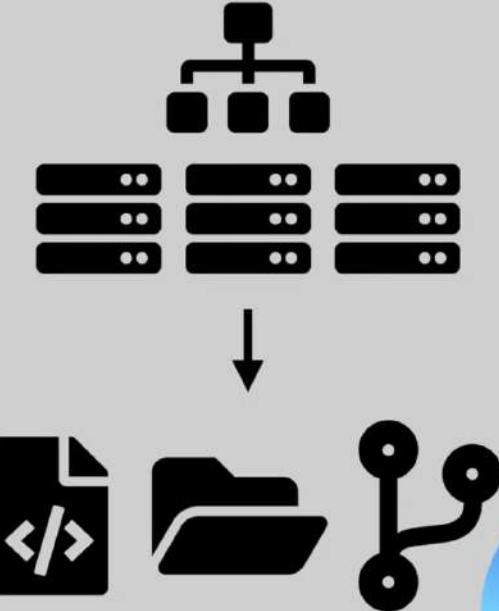
resource "aws_instance" "example" {
  ami           = "ami-08541d54d54812a51" # Ubuntu
  instance_type = "t2.micro"
}
```

IaC



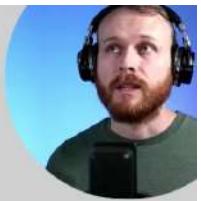
What is Infrastructure as Code (IaC)?

- Categories of IaC tools¹:
 1. Ad hoc scripts
 2. Configuration management tools
 3. Server Templating tools
 4. Orchestration tools
 5. Provisioning tools
- Declarative vs. Imperative

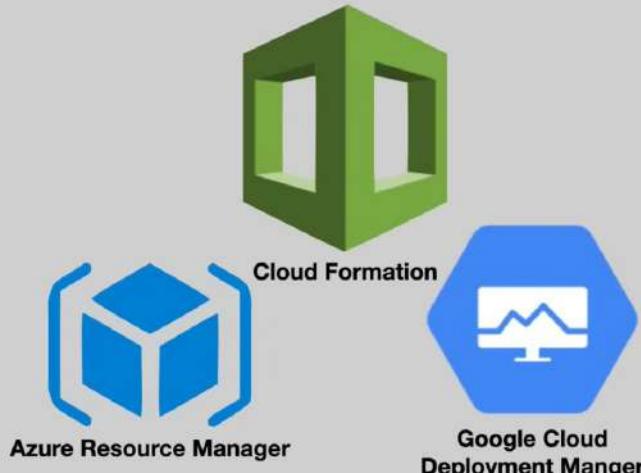


¹ From *Terraform: Up & Running Writing Infrastructure as Code, Second Edition* (O'Reilly Media, 2019) by Yevgeniy Brikman

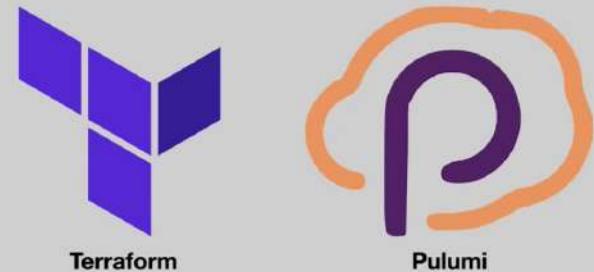
IaC Provisioning Tools Landscape



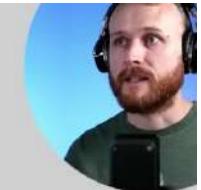
Cloud Specific



Cloud Agnostic



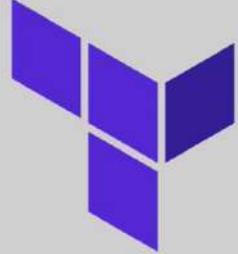
What is Terraform



- Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently
- Enables application software best practices to infrastructure
- Compatible with many clouds and services



Common Patterns

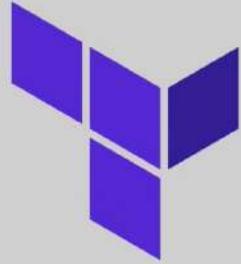


Provisioning



Config Management

Common Patterns

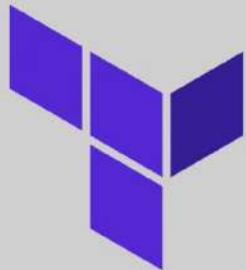


Provisioning



Server Templating

Common Patterns



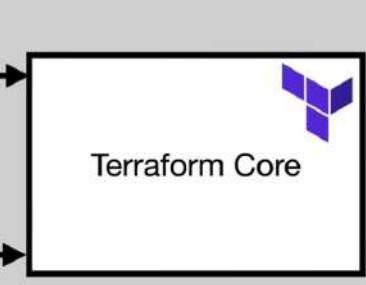
Provisioning



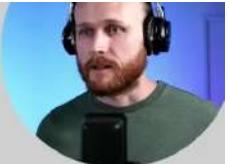
Orchestration

Terraform Architecture

Terraform State



Terraform Config



The screenshot shows the VS Code interface. The Explorer sidebar on the left lists several projects and files under the 'CODE' section, including '.github', '.vscode', '01-cloud-and-iac', '02-overview' (which is currently selected), '03-basics', '04-variables-and-out...', '05-language-features', '06-organization-and...', '07-managing-m...', '08-testing', '09-developer-workflo...', '.gitignore', and 'README.md'. The Editor pane at the top right contains a 'TODO' list with the following items:

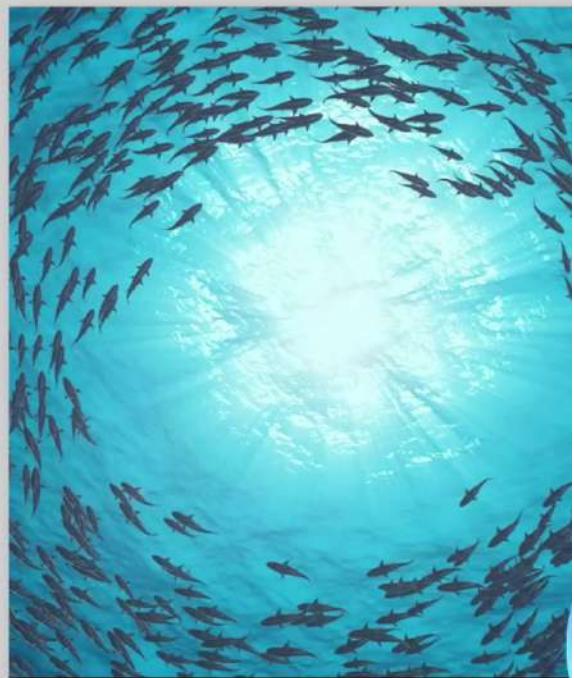
```
1 TODO:  
2   - Install Terraform  
3   - Authenticate to AWS  
4   - Hello world tf config  
5   - Init, Plan, Apply, Destroy
```

The Terminal pane at the bottom shows the command line output:

```
~/development/terraform-course/code main *2 ?3  
~/d/terraform-course/code main *2 ?3  
~/development/terraform-course/code main *2 ?3  
❯
```

Basic Usage Sequence

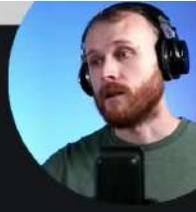
- terraform init
- terraform plan
- terraform apply
- terraform destroy



\$ terraform init



Working Directory



```
$ tree -a .
.
├── .terraform
│   ├── modules
│   │   └── modules.json
│   └── vpc
│       └── main.tf
│           ├── outputs.tf
│           ├── variables.tf
│           ├── versions.tf
│           └── vpc-endpoints.tf
└── providers
    └── registry.terraform.io
        └── hashicorp
            └── aws
                └── 3.23.0
                    └── darwin_amd64
                        └── terraform-provider-aws_v3.23.0_x5
.
└── .terraform.lock.hcl
└── main.tf

39 directories, 102 files
```

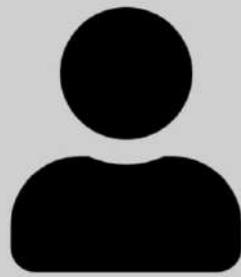
State File

- Terraform's representation of the world
- JSON file containing information about every resource and data object
- Contains Sensitive Info (e.g. database password)
- Can be stored locally or remotely



```
{
  "version": 4,
  "terraform_version": "0.14.4",
  "serial": 5,
  "lineage": "ad1eb9b9-c9a3-e58c-e666-f1ea007e918d",
  "outputs": {},
  "resources": [
    {
      "mode": "managed",
      "type": "aws_instance",
      "name": "example",
      "provider": "provider[\"registry.terraform.io/hashicorp/aws\"]",
      "instances": [
        {
          "schema_version": 1,
          "attributes": {
            "ami": "ami-011899242bb902164",
            "arn": "arn:aws:ec2:us-east-1:917774925227:instance/i-0e9ac03f2e84f846b",
            "public_ip": "3.87.232.28",
            ...
            <MANY MORE ATTRIBUTES>
            ...
          },
          "sensitive_attributes": [],
          "private": <SENSITIVE INFO>
        }
      ]
    }
  ]
}
```

Local Backend



Terraform State

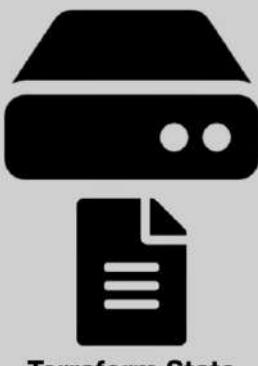
Simple to get started!

Sensitive values in plain text

Uncollaborative

Manual

Remote Backend



Terraform State

Sensitive data encrypted

Collaboration possible

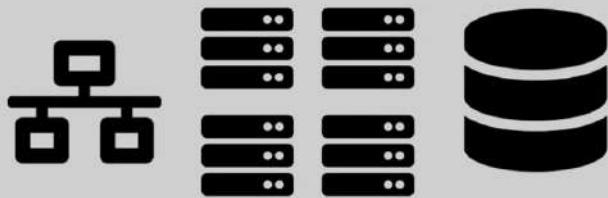
Automation possible

Increased complexity

\$ terraform plan



(Desired State)



Terraform State

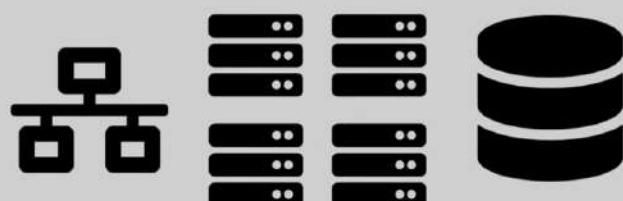


\$ terraform apply



Terraform Config

(Desired State)



Terraform State

(Actual State)



Plan: +1 Virtual Machine



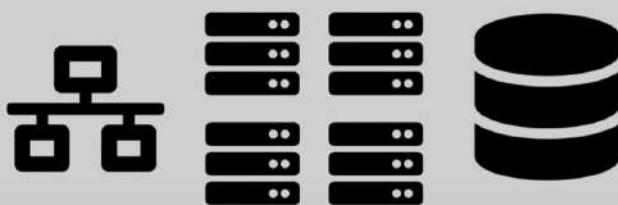
\$ terraform destroy



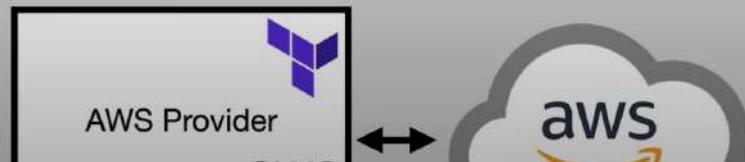
Plan: Destroy Everything



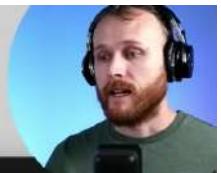
\$ terraform destroy



(Actual State)



Remote Backend (Terraform Cloud)



```
terraform {  
  backend "remote" {  
    organization = "devops-directive"  
  
    workspaces {  
      name = "terraform-course"  
    }  
  }  
}
```

The screenshot shows the Terraform Cloud web interface. At the top, there's a navigation bar with a logo and a dropdown menu. Below it, a header says "devops-directive / Workspaces". A table titled "Workspaces" shows one entry: "terraform-course" with a status of "a few seconds ago". There are buttons for "New workspace" and search. The footer includes the HashiCorp logo and the year 2021.

Free up to 5 users

\$20/user/month beyond that

Remote Backend (AWS)

```
terraform {  
  backend "s3" {  
    bucket      = "devops-directive-tf-state"  
    key         = "tf-infra/terraform.tfstate"  
    region      = "us-east-1"  
    dynamodb_table = "terraform-state-locking"  
    encrypt     = true  
  }  
}
```



S3 Bucket used for storage



DynamoDB used for locking



Remote Backend (AWS)

Bootstrapping – part 1

No Remote Backend Specified (defaults to local)

```
terraform {  
    required_providers {  
        aws = {  
            source  = "hashicorp/aws"  
            version = "~> 3.0"  
        }  
    }  
  
    provider "aws" {  
        region = "us-east-1"  
    }  
  
    resource "aws_s3_bucket" "terraform_state" {  
        bucket      = "devops-directive-tf-state"  
        force_destroy = true  
        versioning {  
            enabled = true  
        }  
  
        server_side_encryption_configuration {  
            rule {  
                apply_server_side_encryption_by_default {  
                    sse_algorithm = "AES256"  
                }  
            }  
        }  
    }  
  
    resource "aws_dynamodb_table" "terraform_locks" {  
        name      = "terraform-state-locking"  
        billing_mode = "PAY_PER_REQUEST"  
        hash_key    = "LockID"  
        attribute {  
            name = "LockID"  
            type = "S"  
        }  
    }  
}
```

Remote Backend (AWS)

Bootstrapping – part 1

Versioned and encrypted S3 Bucket

DynamoDB Table with hash_key = “LockID”

```
resource "aws_s3_bucket" "terraform_state" {  
    bucket      = "devops-directive-tf-state"  
    force_destroy = true  
    versioning {  
        enabled = true  
    }  
  
    server_side_encryption_configuration {  
        rule {  
            apply_server_side_encryption_by_default {  
                sse_algorithm = "AES256"  
            }  
        }  
    }  
}  
  
resource "aws_dynamodb_table" "terraform_locks" {  
    name      = "terraform-state-locking"  
    billing_mode = "PAY_PER_REQUEST"  
    hash_key    = "LockID"  
    attribute {  
        name = "LockID"  
        type = "S"  
    }  
}
```



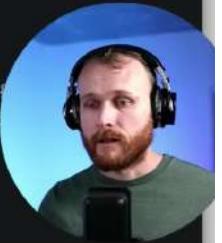
Remote Backend (AWS)

Bootstrapping – part 2

Remote Backend Specified as S3 bucket

Remaining configuration unchanged

```
terraform {  
  backend "s3" {  
    bucket      = "devops-directive-tf-state"  
    key         = "tf-infra/terraform.tfstate"  
    region      = "us-east-1"  
    dynamodb_table = "terraform-state-locking"  
    encrypt     = true  
  }  
  
  required_providers {  
    aws = {  
      source  = "hashicorp/aws"  
      version = "~> 3.0"  
    }  
  }  
  
  provider "aws" {  
    region = "us-east-1"  
  }  
  
  resource "aws_s3_bucket" "terraform_state" {  
    bucket      = "devops-directive-tf-state"  
    force_destroy = true  
    versioning {  
      enabled = true  
    }  
  }  
}
```

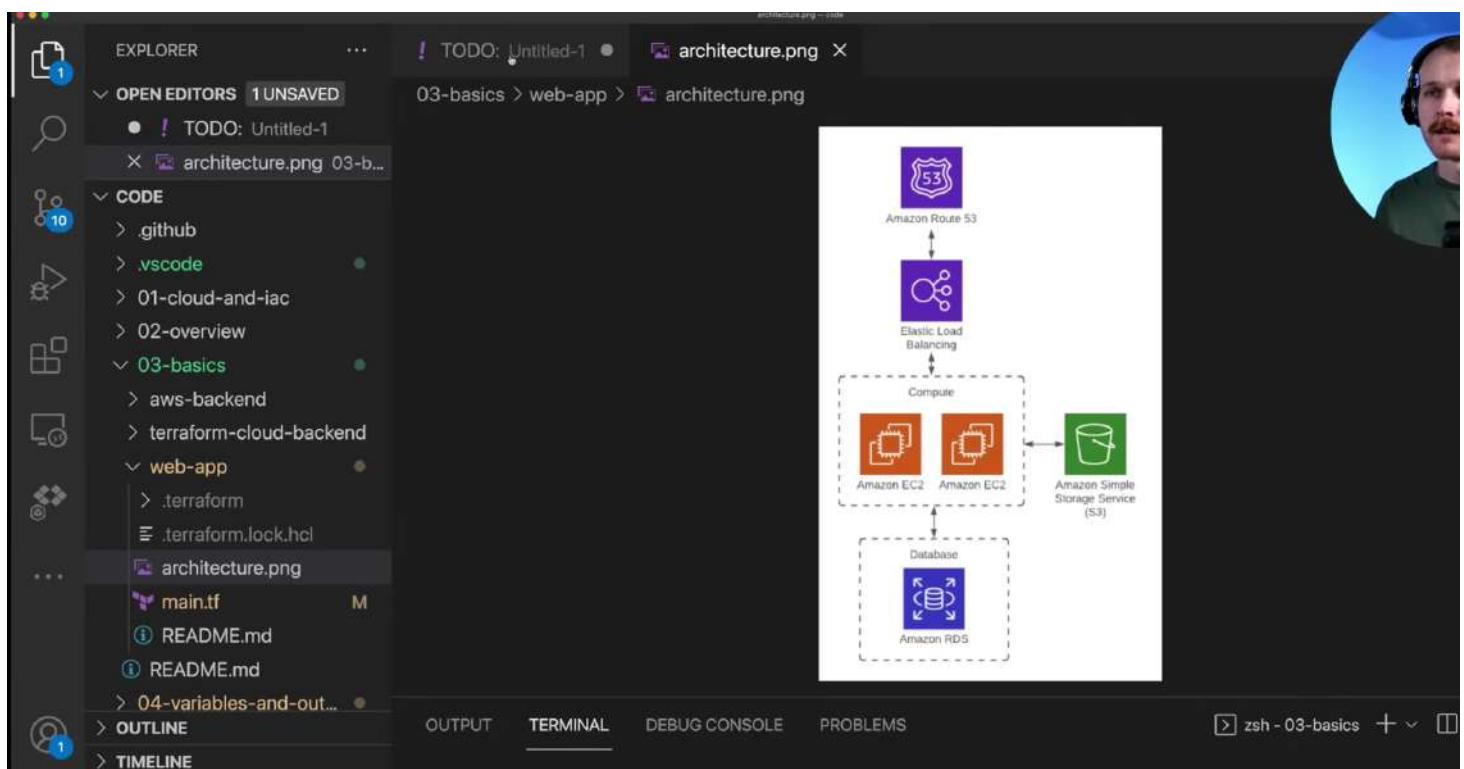
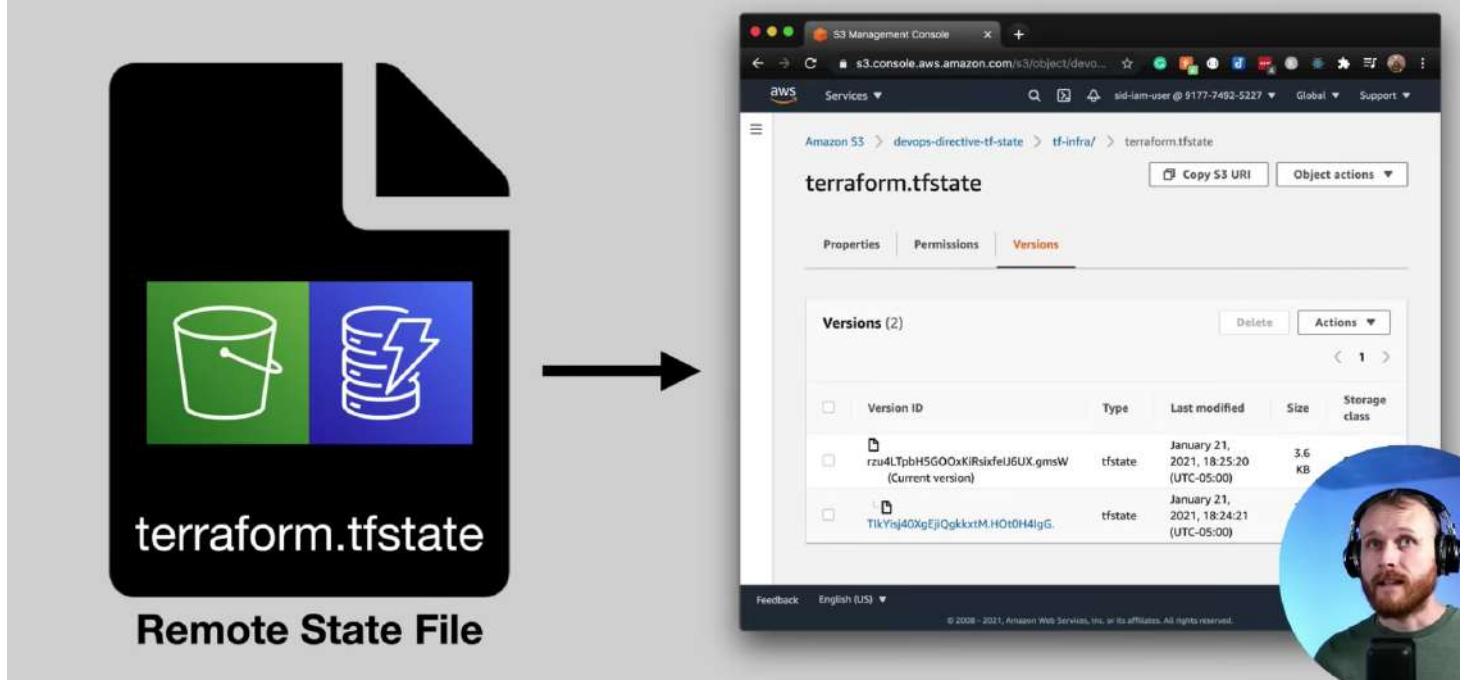


Remote Backend (AWS)

Bootstrapping – part 2

```
$ terraform init  
Initializing the backend...  
Do you want to copy existing state to the new backend?  
Pre-existing state was found while migrating the previous "local" backend to the  
newly configured "s3" backend. No existing state was found in the newly  
configured "s3" backend. Do you want to copy this state to the new "s3"  
backend? Enter "yes" to copy and "no" to start with an empty state.  
Enter a value: yes  
  
Successfully configured the backend "s3"! Terraform will automatically  
use this backend unless the backend configuration changes.  
  
Initializing provider plugins...  
- Reusing previous version of hashicorp/aws from the dependency lock file  
- Installing hashicorp/aws v3.23.0...  
- Installed hashicorp/aws v3.23.0 (signed by HashiCorp)  
  
Terraform has been successfully initialized!  
  
You may now begin working with Terraform. Try running "terraform plan" to see  
any changes that are required for your infrastructure. All Terraform commands  
should now work.  
  
If you ever set or change modules or backend configuration for Terraform,  
rerun this command to reinitialize your working directory. If you forget, other  
commands will detect it and remind you to do so if necessary.  
  
$ terraform plan  
aws_s3_bucket.terraform_state: Refreshing state... [id=devops-directive-tf-state]  
aws_dynamodb_table.terraform_locks: Refreshing state... [id=terraform-state-locking]  
  
No changes. Infrastructure is up-to-date.  
  
This means that Terraform did not detect any differences between your  
configuration and real physical resources that exist. As a result, no  
actions need to be performed.
```

Remote Backend (AWS)



Variable Types

- Input Variables
 - var.<name>

- Local Variables
 - local.<name>

- Output Variables

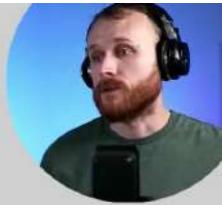
```
variable "instance_type" {  
    description = "ec2 instance type"  
    type        = string  
    default     = "t2.micro"  
}  
  
locals {  
    service_name = "My Service"  
    owner        = "DevOps Directive"  
}  
  
output "instance_ip_addr" {  
    value = aws_instance.instance.public_ip  
}
```

Setting Input Variables

(In order of precedence // lowest → highest)

- Manual entry during plan/apply
- Default value in declaration block
- TF_VAR_<name> environment variables
- terraform.tfvars file
- *.auto.tfvars file
- Command line -var or -var-file

Types & Validation



Primitive Types:

- string
- number
- bool

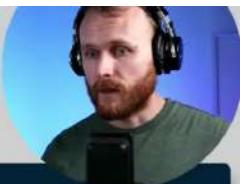
Complex Types:

- list(<TYPE>)
- set(<TYPE>)
- map(<TYPE>)
- object({<ATTR NAME> = <TYPE>, ... })
- tuple([<TYPE>, ...])

Validation:

- Type checking happens automatically
- Custom conditions can also be enforced

Sensitive Data



Mark variables as sensitive:

- Sensitive = true

Pass to terraform apply with:

- TV_VAR_variable
- -var (retrieved from secret manager at runtime)

Can also use external secret store

- For example, AWS Secrets Manager

```
● ● ●

Terraform will perform the following actions:

  # some_resource.a will be created
  + resource "some_resource" "a" {
      + sensitive_value  = (sensitive)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

Part 5

Additional Language Features



Expressions + Functions

Use the docs!

Expressions

- Template strings
- Operators (!, -, *, /, %, >, ==, etc...)
- Conditionals (cond ? true : false)
- For (**[for o in var.list : o.id]**)
- Splat (var.list[*].id)
- Dynamic Blocks
- Constraints (Type & Version)

Functions

- Numeric
- String
- Collection
- Encoding
- Filesystem
- Date & Time
- Hash & Crypto
- IP Network
- Type Conversion



Meta-Arguments

depends_on

- Terraform automatically generates dependency graph based on references
- If two resources depend on each other (but not each others data), *depends_on* specifies that dependency to enforce ordering
- For example, if software on the instance needs access to S3, trying to create the `aws_instance` would fail if attempting to create it before the `aws_iam_role_policy`

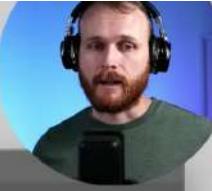
A screenshot of a terminal window titled "main.tf". The code defines three resources: an IAM role, an IAM instance profile, and an IAM role policy. The IAM role policy resource includes a "depends_on" block that points to the IAM role and the IAM instance profile resources. An orange rectangle highlights this "depends_on" block. In the top right corner of the terminal window, there is a circular profile picture of the same man with a beard and headphones from the previous slide.

```
1 resource "aws_iam_role" "example" {
2   name = "example"
3   assume_role_policy = "..."
4 }
5
6 resource "aws_iam_instance_profile" "example" {
7   role = aws_iam_role.example.name
8 }
9
10 resource "aws_iam_role_policy" "example" {
11   name = "example"
12   role = aws_iam_role.example.name
13   policy = jsonencode({
14     "Statement" = [
15       "Action" = "s3:*",
16       "Effect" = "Allow",
17     ],
18   })
19 }
20
21 resource "aws_instance" "example" {
22   ami           = "ami-a1b2c3d4"
23   instance_type = "t2.micro"
24
25   iam_instance_profile = aws_iam_instance_profile.example
26
27 depends_on = [
28   aws_iam_role_policy.example,
29 ]
30 }
```

Meta-Arguments

Count

- Allows for creation of multiple resources/modules from a single block
- Useful when the multiple necessary resources are nearly identical



```
main.tf
Users > palas > Desktop > main.tf
1 resource "aws_instance" "server" {
2   count = 4 # create four EC2 instances
3
4   ami           = "ami-a1b2c3d4"
5   instance_type = "t2.micro"
6
7   tags = {
8     Name = "Server ${count.index}"
9   }
10 }
```

Meta-Arguments

for_each

- Allows for creation of multiple resources/modules from a single block
- Allows more control to customize each resource than *count*

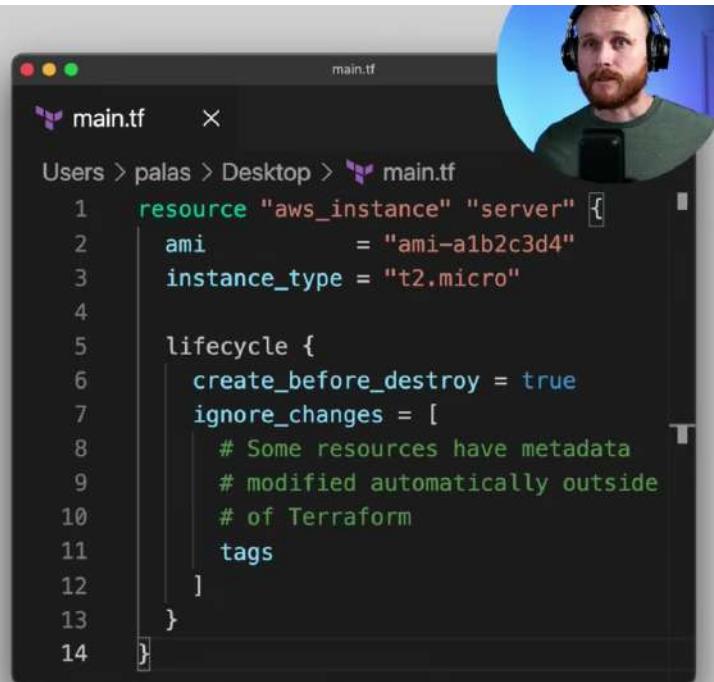


```
main.tf
Users > palas > Desktop > main.tf
1 locals {
2   subnet_ids = toset([
3     "subnet-abcdef",
4     "subnet-012345",
5   ])
6 }
7
8 resource "aws_instance" "server" {
9   for_each = local.subnet_ids
10
11   ami           = "ami-a1b2c3d4"
12   instance_type = "t2.micro"
13   subnet_id    = each.key
14
15   tags = {
16     Name = "Server ${each.key}"
17   }
18 }
```

Meta-Arguments

Lifecycle

- A set of meta arguments to control terraform behavior for specific resources
- *create_before_destroy* can help with zero downtime deployments
- *ignore_changes* prevents Terraform from trying to revert metadata being set elsewhere
- *prevent_destroy* causes Terraform to reject any plan which would destroy this resource

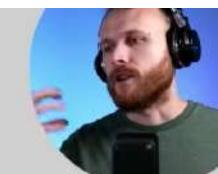
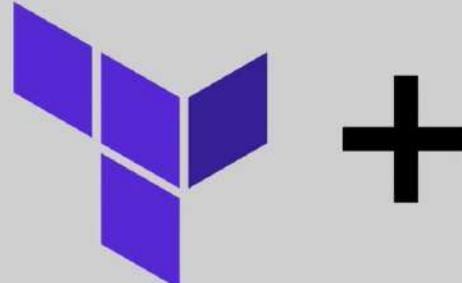


```
main.tf
Users > palas > Desktop > main.tf
1 resource "aws_instance" "server" {
2   ami           = "ami-a1b2c3d4"
3   instance_type = "t2.micro"
4
5   lifecycle {
6     create_before_destroy = true
7     ignore_changes = [
8       # Some resources have metadata
9       # modified automatically outside
10      # of Terraform
11      tags
12    ]
13  }
14}
```

Provisioners

Perform action on local or remote machine

- file
- local-exec
- remote-exec
- vendor
 - chef
 - puppet



Ansible



Chef



Puppet

Hashicorp Configuration Language Features

The official documentation is the best reference for these: <https://www.terraform.io/docs/language/index.html>

NOTE: ````py` is used on code blocks to get highlighting since HCL isn't an allowable language.

Expressions

Strings

```
"foo" # literal string  
"foo ${var.bar}" # template string
```

Operators

```
# Order of operations:  
*, ~ # (multiplication by -1)  
*, /, % # (modulo)  
+, - # (subtraction)  
>, >=, <, <= # (comparison)  
==, != # (equality)  
&& # (AND)  
|| # (OR)
```

Part 6

Project Organization + Modules



What is a Module?

Modules are containers for multiple resources that are used together. A module consists of a collection of .tf and/or .tf.json files kept together in a directory.

Modules are the main way to package and reuse resource configurations with Terraform.

Types of Modules



- **Root Module:** Default module containing all .tf files in main working directory
- **Child Module:** A separate external module referred to from a .tf file

Module Sources:

- Local paths
- Terraform Registry
- GitHub
- Bitbucket
- Generic Git, Mercurial repositories
- HTTP URLs
- S3 buckets
- GCS buckets

Module Sources



- **Local Path**

```
● ● ●  
module "web-app" {  
  source = "../web-app"  
}
```

- **Terraform Registry**

```
● ● ●  
module "consul" {  
  source = "hashicorp/consul/aws"  
  version = "0.1.0"  
}
```

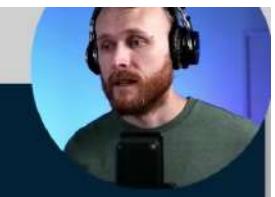
Module Sources

- **Github:**
 - **HTTPS** →
 - **SSH** →
- **Generic Git Repo**

```
# HTTPS
module "example" {
  source = "github.com/hashicorp/example?ref=v1.2.0"
}

# SSH
module "example" {
  source = "git@github.com:hashicorp/example.git"
}

# GENERIC
module "example" {
  source = "git::ssh://username@example.com/storage.git"
}
```



Inputs + Meta-arguments

- Input variables are passed in via module block

Meta-Arguments:

- count
- for_each
- providers
- depends_on

```
module "web_app" {
  source = "../web-app-module"

  # Input Variables
  bucket_name = "devops-directive-web-app-data"
  domain      = "mysuperawesomesite.com"
  db_name     = "mydb"
  db_user     = "foo"
  db_pass     = var.db_pass
}
```



What Makes a Good Module?

- Raises the abstraction level from base resource types
- Groups resources in a logical fashion
- Exposes input variables to allow necessary customization + composition
- Provides useful defaults
- Returns outputs to make further integrations possible



Terraform registry

A screenshot of the Terraform Registry website. The main navigation bar includes 'Terraform', 'Registry', 'Search Providers and Modules', 'Browse', 'Publish', and 'Sign-In'. Below the navigation, there's a search bar and a 'Providers' dropdown menu. The main content area is titled '# Modules' and contains a list of modules. One module, 'aws/security-group', is highlighted with a red box. Other visible modules include 'terraform-aws-modules/vpc', 'terraform-aws-modules/eks', 'terraform-aws-modules/rds', and 'terraform-aws-modules/iam'.

A screenshot of the Terraform Registry showing the details for the 'aws/security-group' module. The page includes a 'Provider' section for AWS, a 'Version 4.0.0 (latest)' section, and a 'Provision Instructions' section with a code snippet:

```
module "security-group" {
  source = "terraform-aws-modules/aws-security-group/main"
  # Insert the 2 regular
```

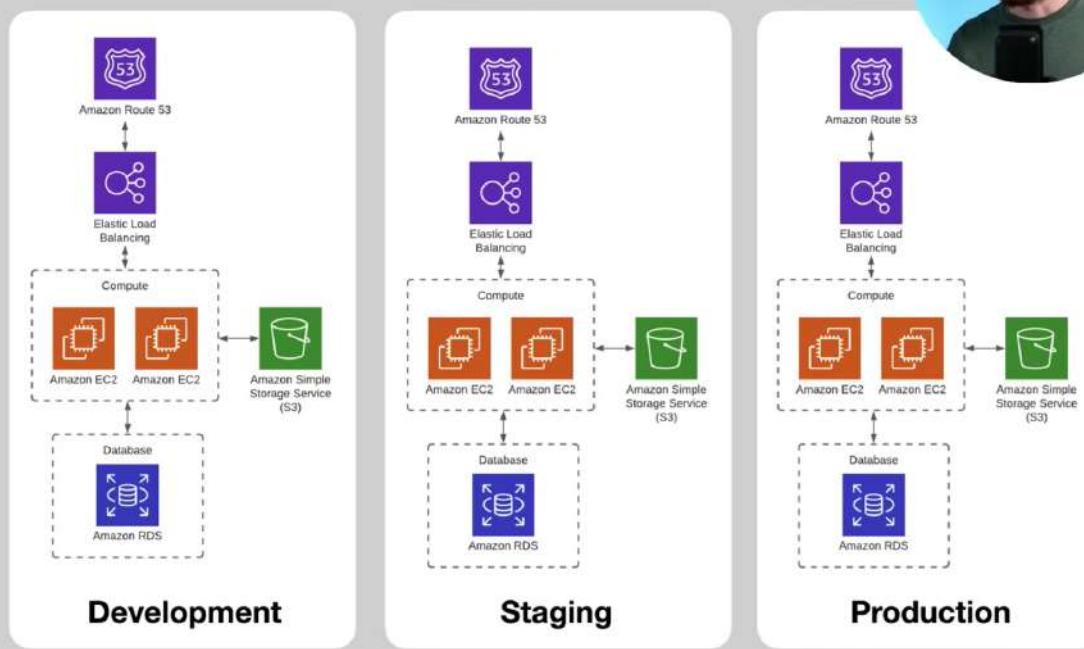
The page also lists 'Inputs (45)', 'Outputs (0)', 'Dependency (1)', and 'Resources (22)'.

Part 7

Managing Multiple Environments



One Config → Multiple Environments



Two Main Approaches

Workspaces

- Multiple named sections within a single backend

```
palex@Cassidy-MacBook-Pro:~ % terraform workspace new dev
Created and switched to workspace "dev"!

You're now on a new, empty workspace. Workspaces isolate their state,
so if you run "terraform plan" Terraform will not see any existing state
for this configuration.
% ~ terraform workspace list
  default
* dev
  production
  staging
% ~ %
```

File Structure

- Directory layout provides separation, modules provide reuse

```
tf-file-structure tree
.
+-- _modules
|   +-- module-1
|   |   |-- main.tf
|   |   |-- variables.tf
|   +-- module-2
|       |-- main.tf
|       |-- variables.tf
+-- dev
    |-- main.tf
    |-- terraform.tfvars
+-- production
    |-- main.tf
    |-- terraform.tfvars
+-- staging
    |-- main.tf
    |-- terraform.tfvars

6 directories, 10 files
```

Terraform Workspaces

```
git:(master) [MacBook-Pro: ~] ~ terraform workspace new dev  
Created and switched to workspace "dev"  
You're now on a new, empty workspace. Workspaces isolate their state,  
so if you run "terraform plan" Terraform will not see any existing state.  
+ terraform workspace list  
  default  
* dev  
  production  
  staging
```

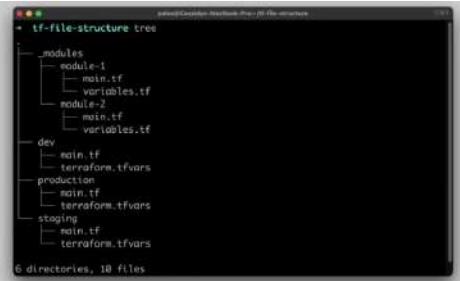
Pros

- Easy to get started
- Convenient `terraform.workspace` expression
- Minimizes Code Duplication

Cons

- Prone to human error
- State stored within same backend
- Codebase doesn't unambiguously show deployment configurations

File Structure

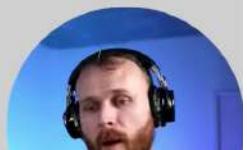


Pros

- Isolation of backends
 - Improved security
- Decreased potential for human error
- Codebase fully represents deployed state

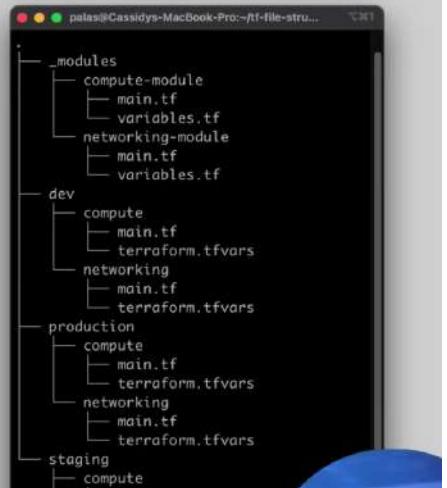
Cons

- Multiple `terraform apply` required to provision environments
- More code duplication, but can be minimized with modules!



File Structure (environments + components)

- Further separation (at logical component groups) useful for larger projects
 - Isolate things that change frequently from those which don't
- Referencing resources across configurations is possible using *terraform_remote_state*



Terragrunt

- Tool by [gruntwork.io](#) that provides utilities to make certain Terraform use cases easier
 - Keeping Terraform code DRY
 - Executing commands across multiple TF configs
 - Working with multiple cloud accounts

Terragrunt
by Gruntwork.io



A screenshot of the Visual Studio Code interface. The top menu bar includes Code, File, Edit, Selection, View, Go, Run, Terminal, Window, and Help. The title bar shows "TODO: Untitled-1 — code". The left sidebar has icons for Explorer, Open Editors (1 unsaved), Code, Search, Problems (11), Run, and others. The Explorer view shows a folder structure with ".github", ".vscode", "01-cloud-and-iac", "02-overview", "03-basics", "04-variables-and-out...", "05-language-features", "06-organization-and...", "07-managing-multipl...", "file-structure", and "workspaces". Inside "workspaces", there are ".terraform", ".terraform.lock.hcl", "main.tf" (marked with a purple icon), and "README.md". The main editor area displays a todo list titled "TODO: Untitled-1" with the following items:

```
1 TODO:  
2 - Managing environments  
3   - Show workspaces approach  
4     - terraform workspace new production  
5     - terraform workspace list  
6     - terraform workspace select staging  
7   - Show directory structure approach  
8     - some "global" resources  
9   - Isolated AWS accounts
```

Part 8

Testing Terraform Code

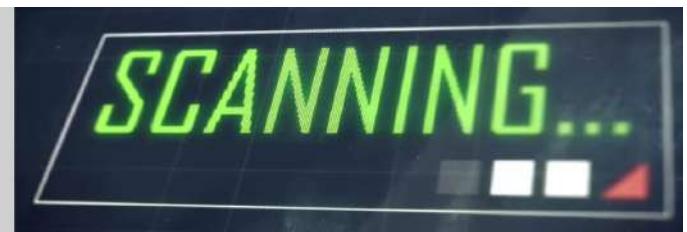


Code Rot

- Out of band changes
- Unpinned versions
- Deprecated dependencies
- Unapplied changes



Static Checks



Built in

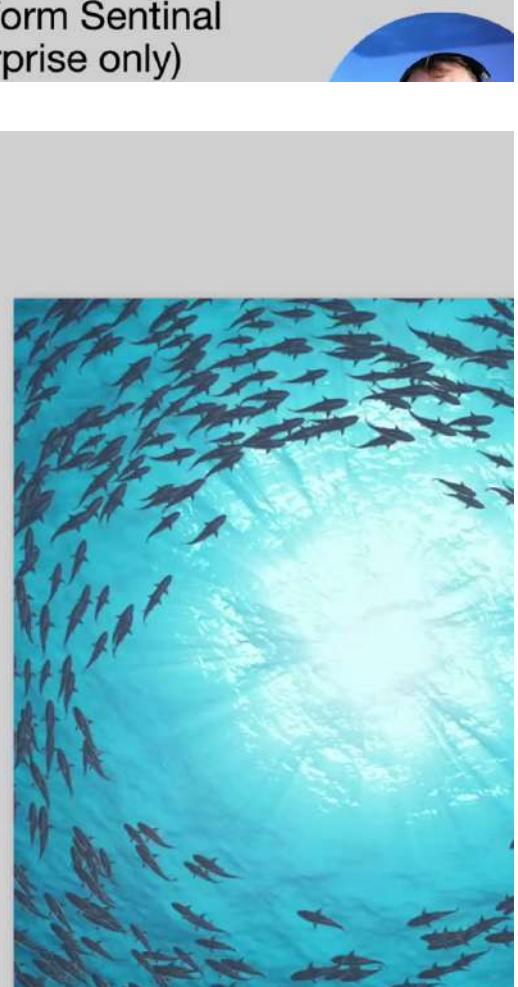
- terraform fmt
- terraform validate
- terraform plan
- custom validation rules

External

- tflint
- checkov, tfsec, terrascan, terraform-compliance, snyk
- Terraform Sentinel (enterprise only)

Manual Testing

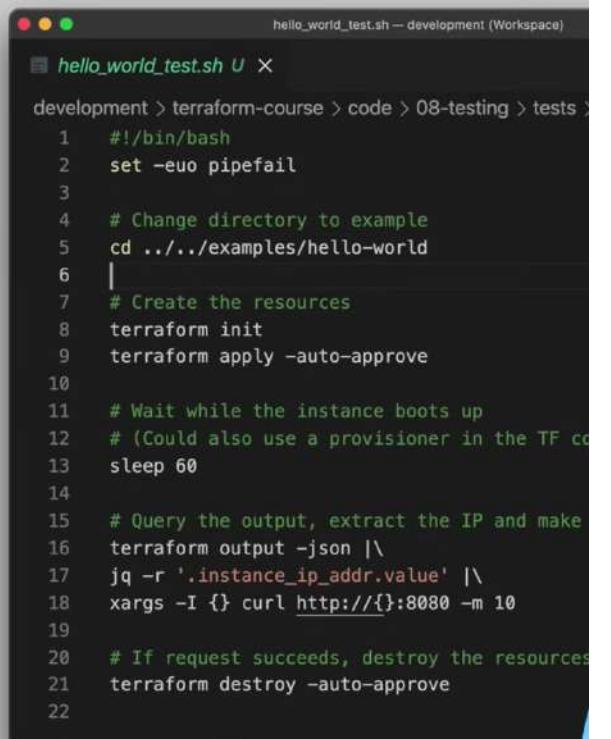
- terraform init
- terraform plan
- terraform apply
- terraform destroy



Automated Testing

Automate the manual test steps...

...with Bash!

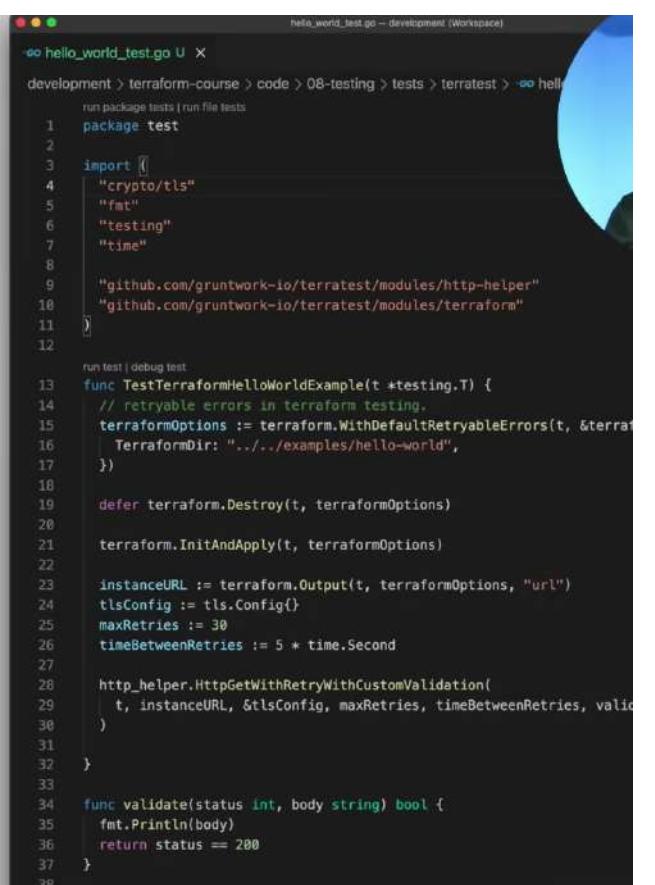


```
hello_world_test.sh U X
development > terraform-course > code > 08-testing > tests >
1  #!/bin/bash
2  set -euo pipefail
3
4  # Change directory to example
5  cd ../../examples/hello-world
6  |
7  # Create the resources
8  terraform init
9  terraform apply --auto-approve
10
11 # Wait while the instance boots up
12 # (Could also use a provisioner in the TF config)
13 sleep 60
14
15 # Query the output, extract the IP and make
16 terraform output -json | \
17 jq -r '.instance_ip_addr.value' | \
18 xargs -I {} curl http://{}:8080 -m 10
19
20 # If request succeeds, destroy the resources
21 terraform destroy --auto-approve
22
```

Automated Testing

Automate the manual test steps...

...with Terratest!



```
hello_world_test.go U X
development > terraform-course > code > 08-testing > tests > terratest > >> hello_world_test.go
run package tests | run file tests
1 package test
2
3 import [
4     "crypto/tls"
5     "fmt"
6     "testing"
7     "time"
8
9     "github.com/gruntwork-io/terratest/modules/http-helper"
10    "github.com/gruntwork-io/terratest/modules/terraform"
11 ]
12
13 run test | debug test
14 func TestTerraformHelloWorldExample(t *testing.T) {
15     // retryable errors in terraform testing.
16     terraformOptions := terraform.WithDefaultRetryableErrors(t, &terraform.Options{
17         TerraformDir: "../../examples/hello-world",
18     })
19     defer terraform.Destroy(t, terraformOptions)
20
21     terraform.InitAndApply(t, terraformOptions)
22
23     instanceURL := terraform.Output(t, terraformOptions, "url")
24     tlsConfig := tls.Config{}
25     maxRetries := 30
26     timeBetweenRetries := 5 * time.Second
27
28     http_helper.HttpGetWithRetryWithCustomValidation(
29         t, instanceURL, &tlsConfig, maxRetries, timeBetweenRetries, validateResponse)
30     }
31 }
32
33 func validate(status int, body string) bool {
34     fmt.Println(body)
35     return status == 200
36 }
37
38
```

