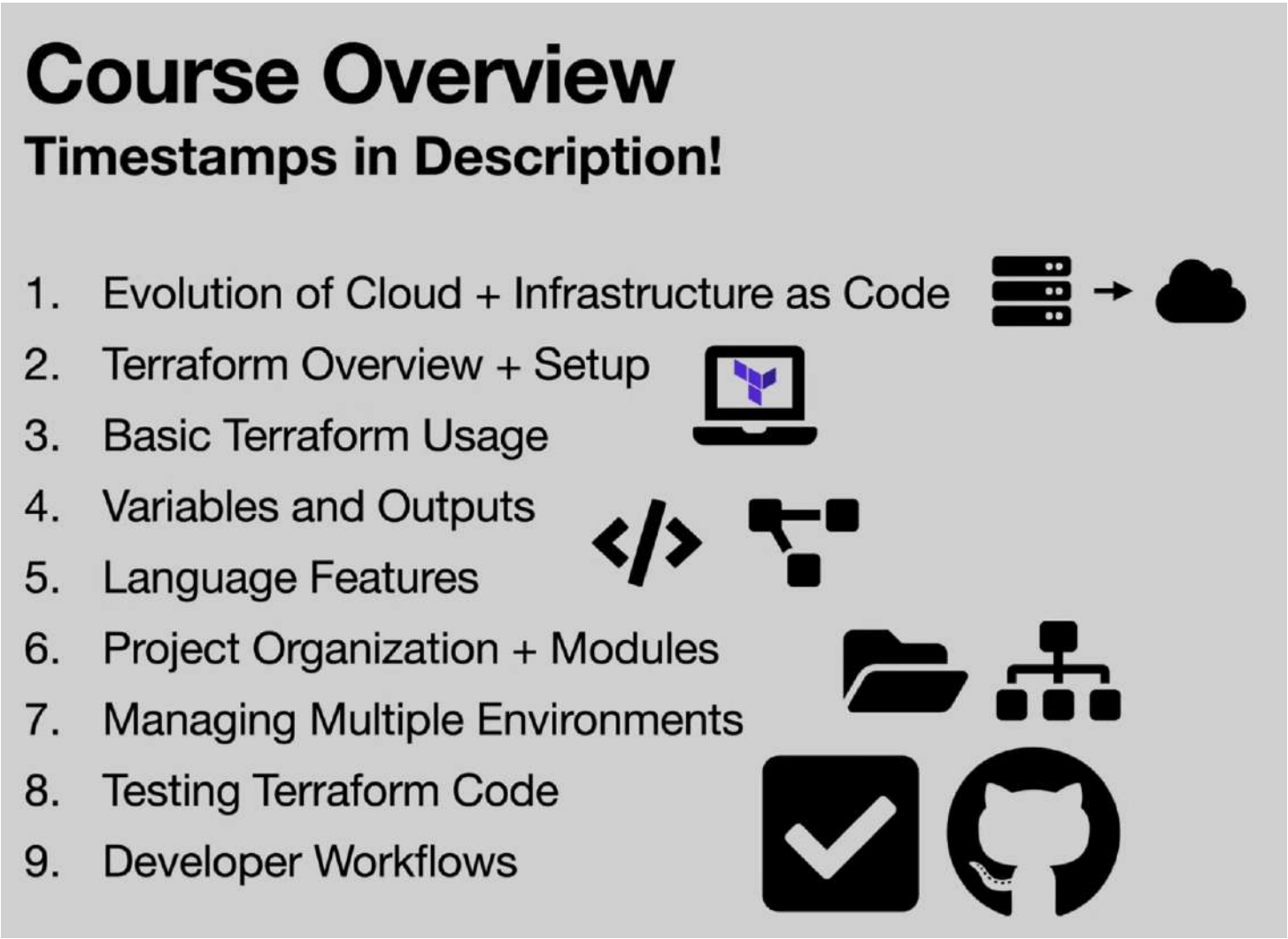
- Terraform Language Documentation

Courses:

- Terraform Course - Automate your AWS cloud infrastructure

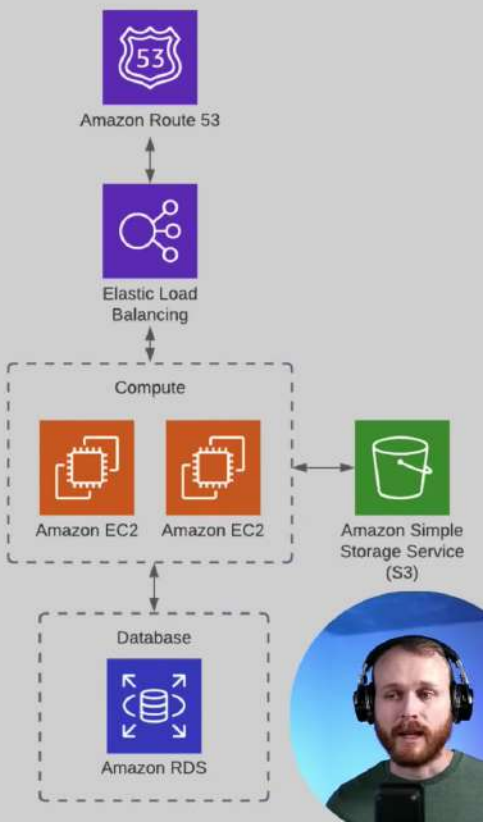
Complete Terraform Course - From BEGINNER to PRO! (Learn Infrastructure as Code)

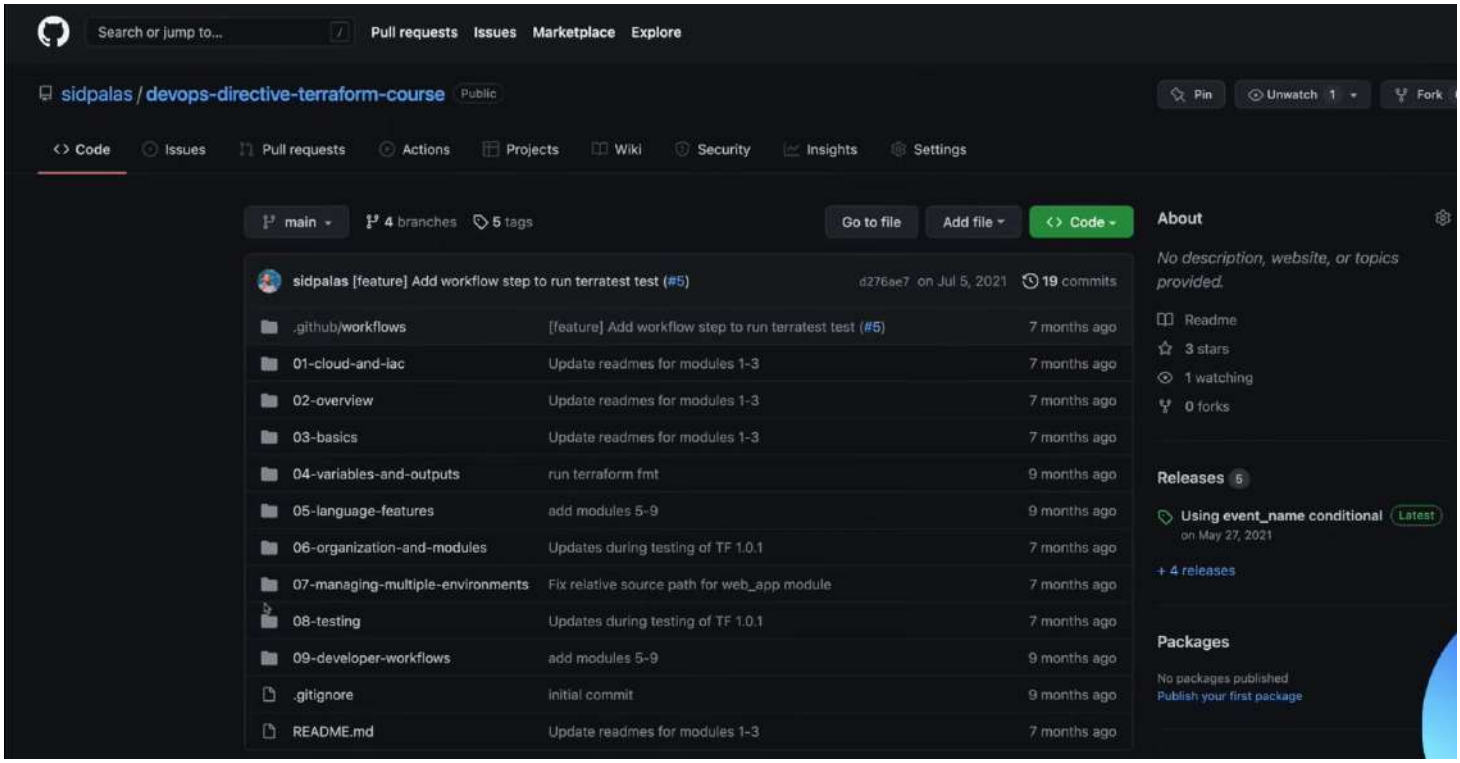
- `terraform apply -var-file=file_name`
- `terraform apply -var="db_user=myuser" -var="db_pass=secretpassword"`
- `$ terraform workspace list`
- `$ terraform workspace new prodaction`



Reference Architecture

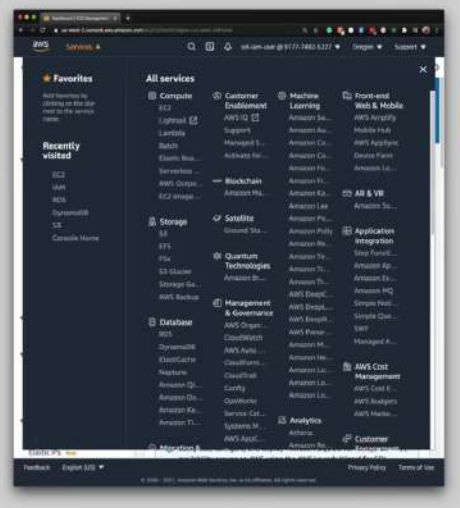
- Basic web application
- Infrastructure all within AWS
- Multiple instances running on EC2
- Using default VPC for simplicity

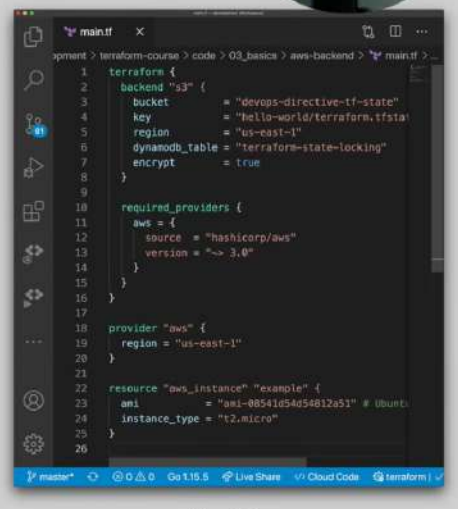




Provisioning Cloud Resources

Three Approaches

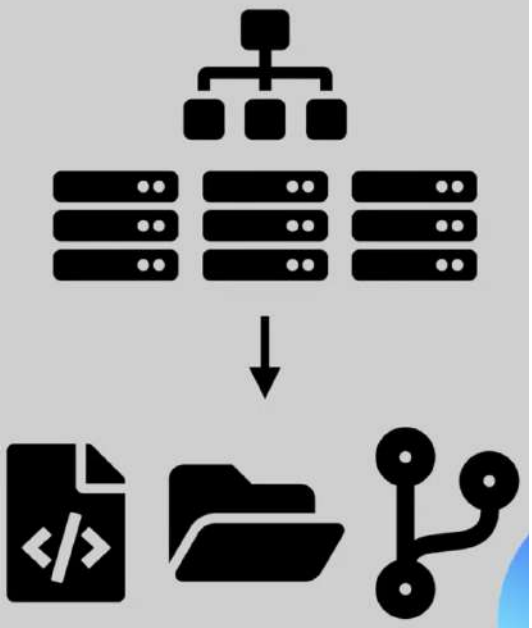




GUI API/CLI IaC

What is Infrastructure as Code (IaC)?

- Categories of IaC tools[1]:
  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 Formation

Azure Resource Manager

Google Cloud Deployment Manger

Cloud Agnostic

Terraform

Pulumi

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

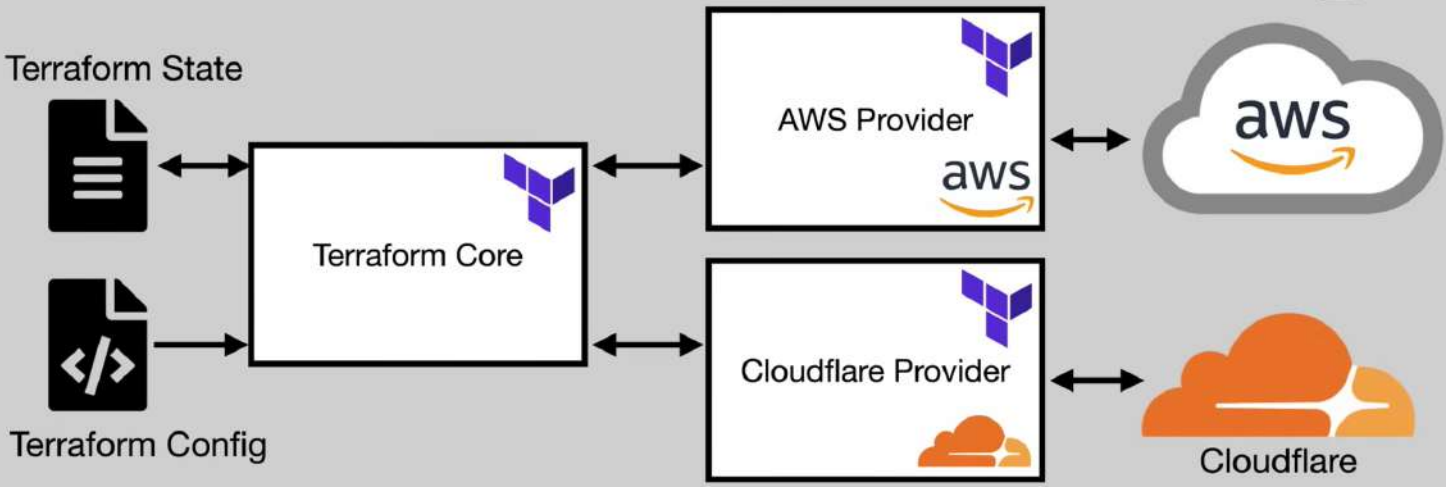
Common Patterns

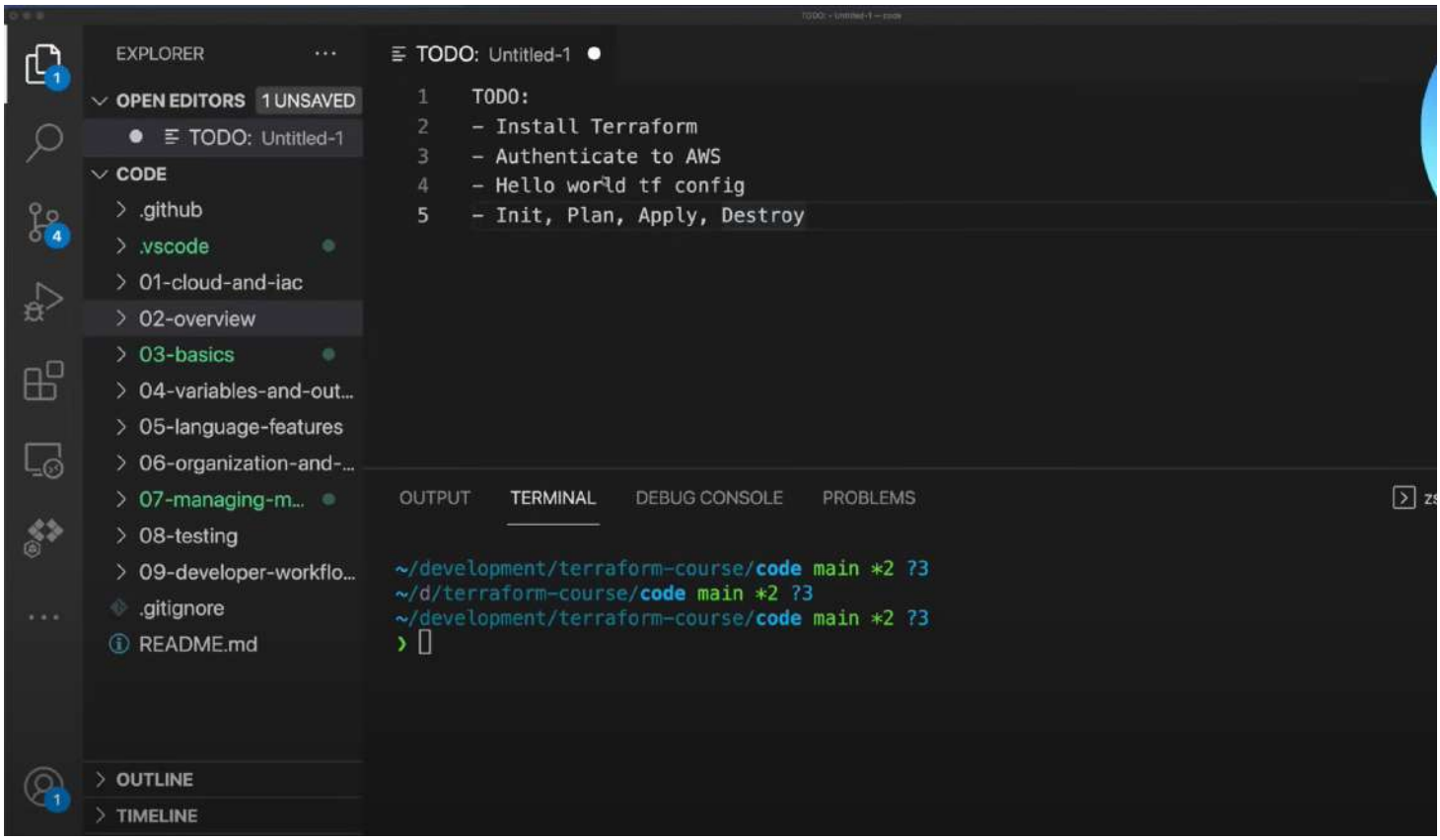
Provisioning

+

Orchestration

Terraform Architecture





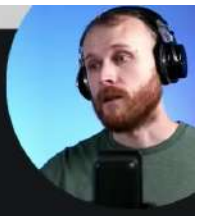
Basic Usage Sequence

- 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
│   │       └── vpc-flow-logs.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
```

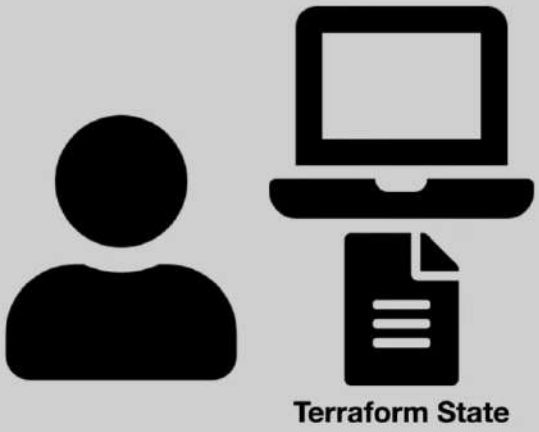


State File

- 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

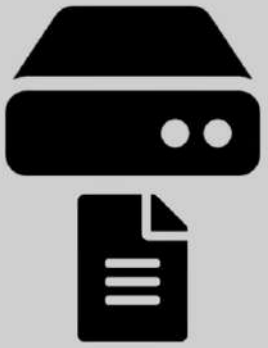
```
{
  "version": 4,
  "terraform_version": "0.14.4",
  "serial": 5,
  "lineage": "ad1eb9b9-c9a3-e58c-e666-f1ea007e918d",
  "outputs": {},
  "resources": [
    {
      "mode": "managed",
      "type": "aws_instance",
      "name": "example",
      "provider": "provider[\"registry.terraform.io/hashicorp/aws\"]",
      "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

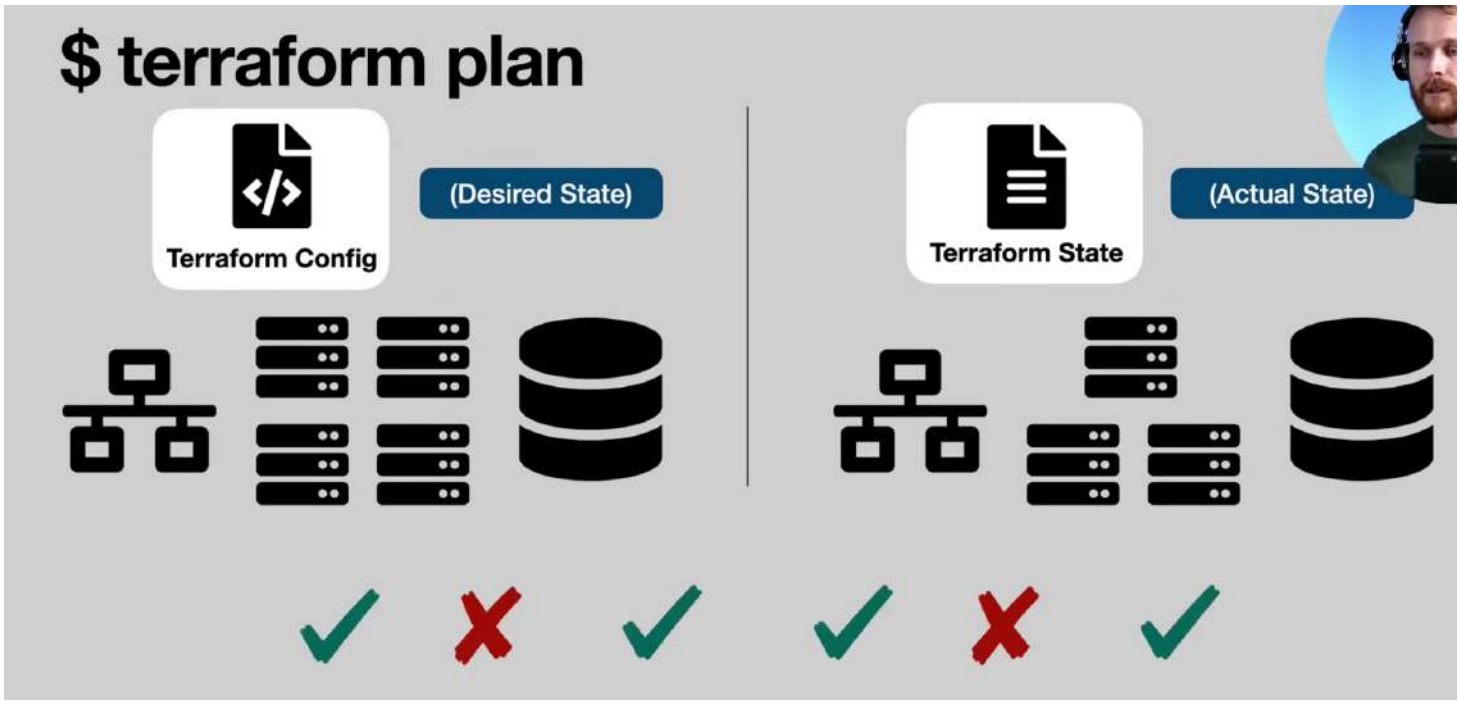


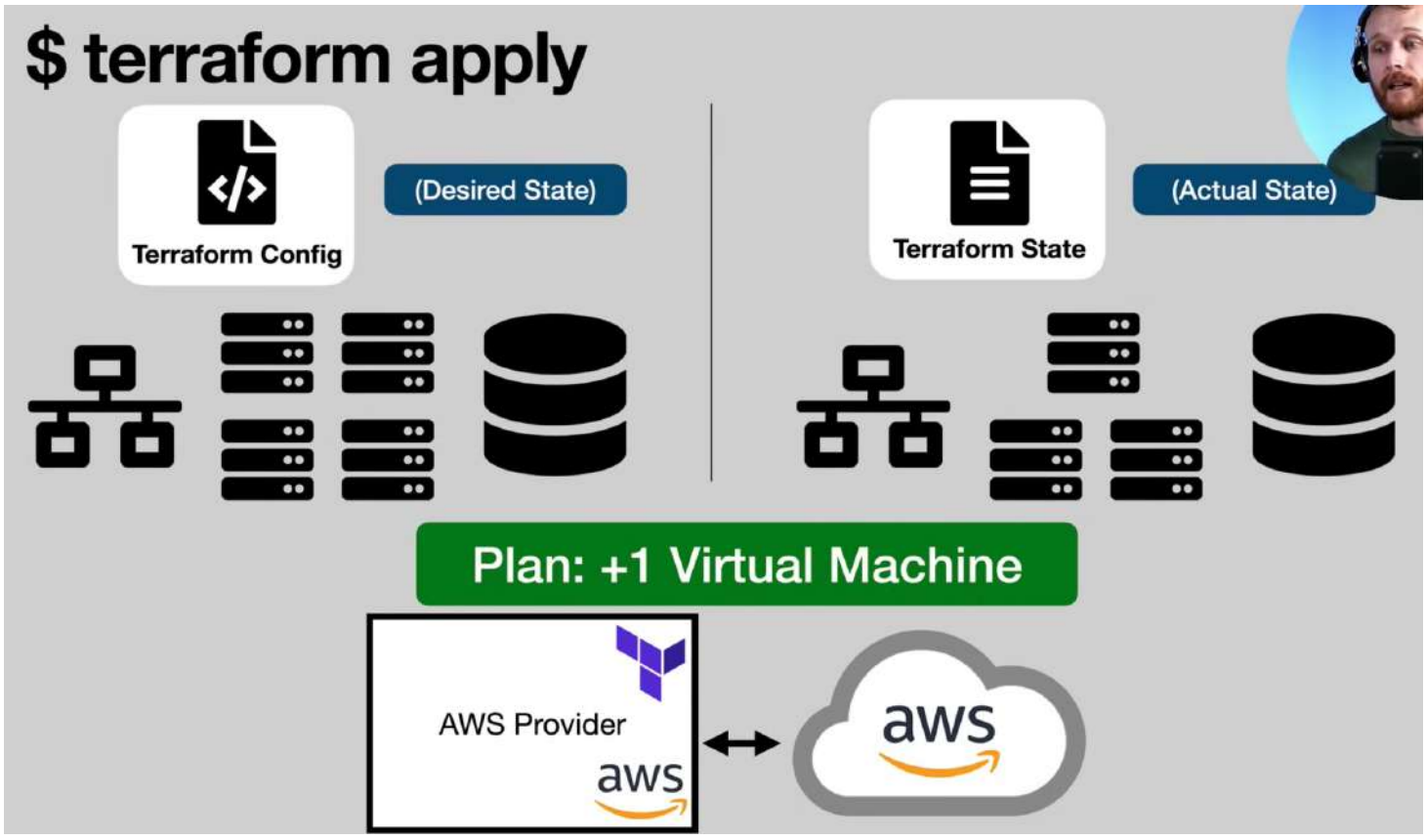
- Simple to get started!
- Sensitive values in plain text
- Uncollaborative
- Manual

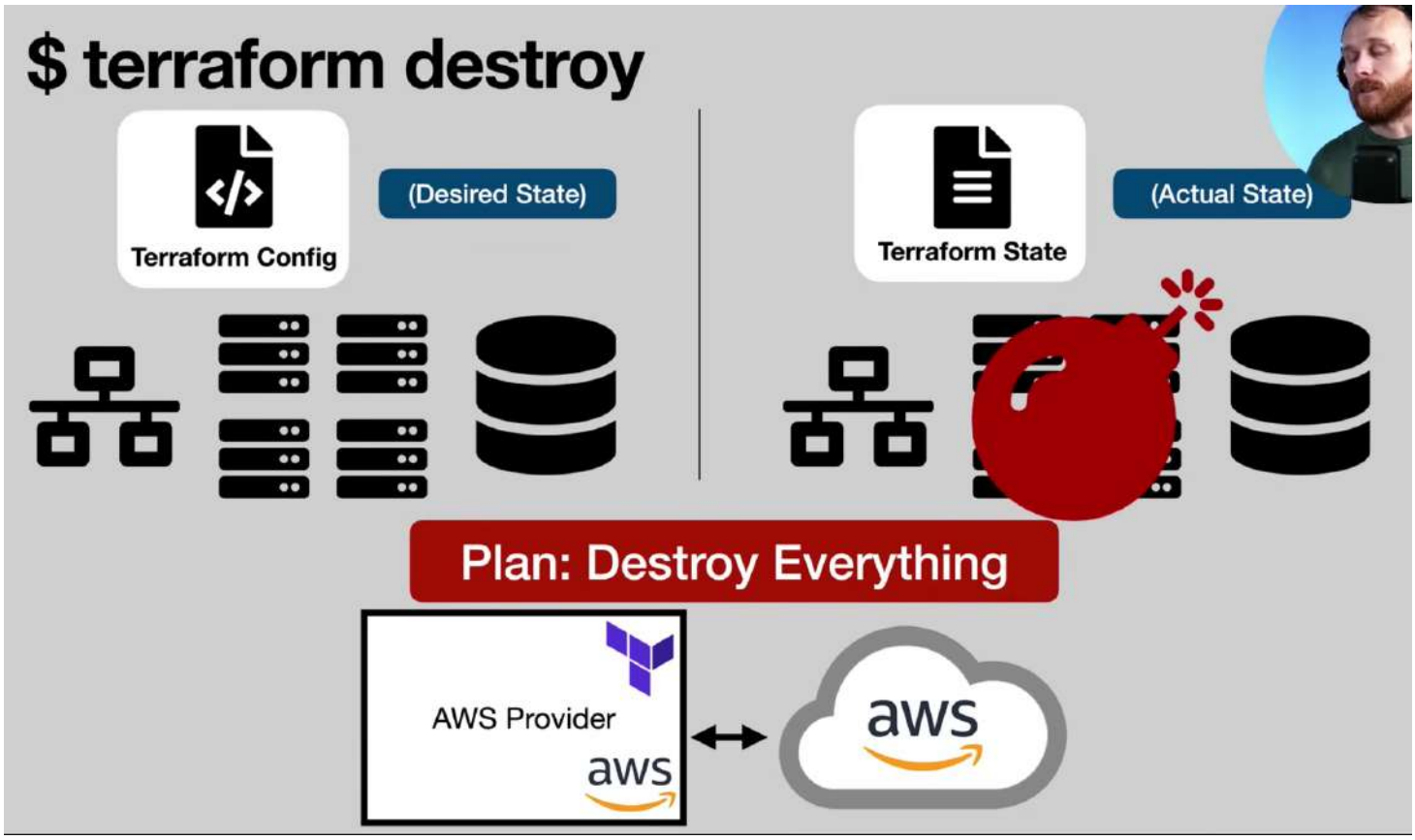
Remote Backend



- Sensitive data encrypted
- Collaboration possible
- Automation possible
- Increased complexity

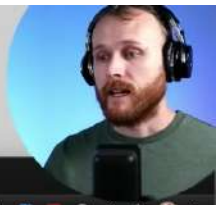

$ terraform plan
Terraform Config
(Desired State)
Terraform State
(Actual State)


$ terraform apply
Terraform Config
(Desired State)
Terraform State
(Actual State)
Plan: +1 Virtual Machine
AWS Provider
aws
aws


$ terraform destroy
Terraform Config
(Desired State)
Terraform State
(Actual State)
Plan: Destroy Everything
AWS Provider
aws
aws


$ terraform destroy
Terraform Config
(Desired State)
Terraform State
(Actual State)
AWS Provider
aws

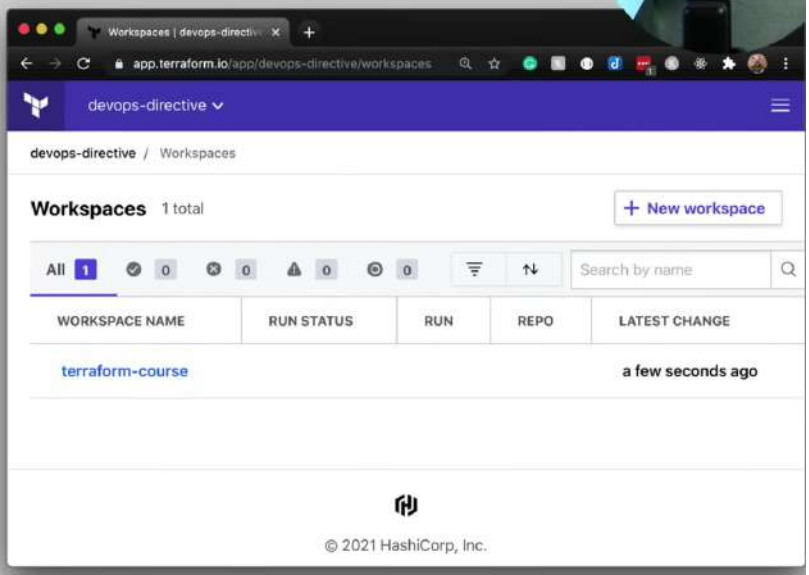
Remote Backend (Terraform Cloud)



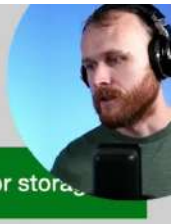
```
terraform {
  backend "remote" {
    organization = "devops-directive"

    workspaces {
      name = "terraform-course"
    }
  }
}
```

Free up to 5 users

$20/user/month beyond that



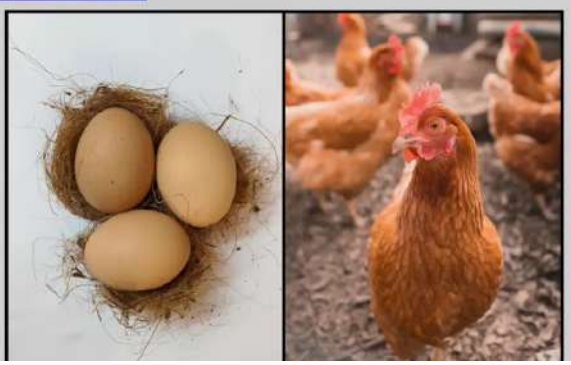
Remote Backend (AWS)



```
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 stora

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_
  bucket        = "devops-directive-
  force_destroy = true
  versioning {
```

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"
  }
}
```

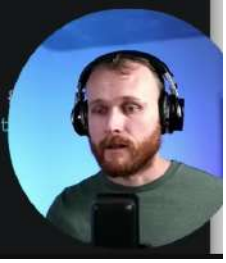
Remote Backend (AWS)

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-stat
  force_destroy = true
  versioning {
    enabled = true
  }
```

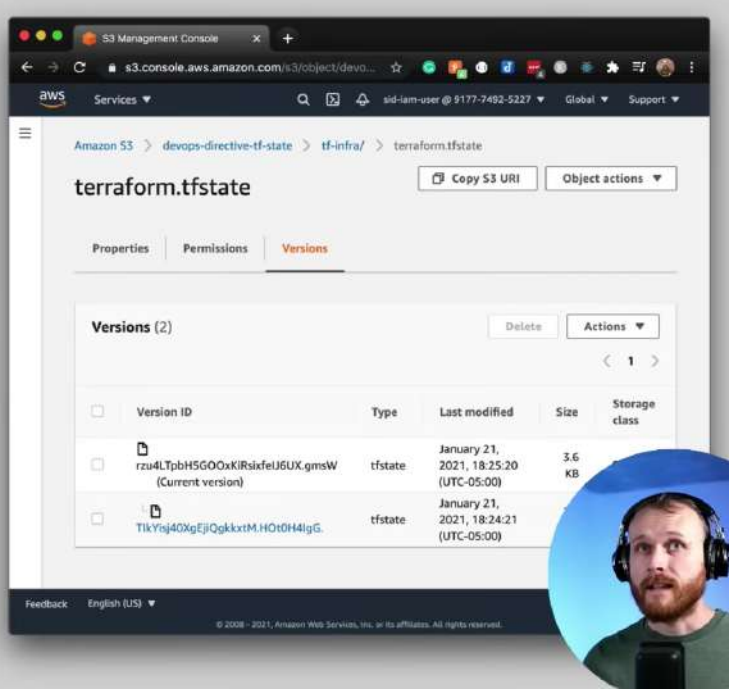


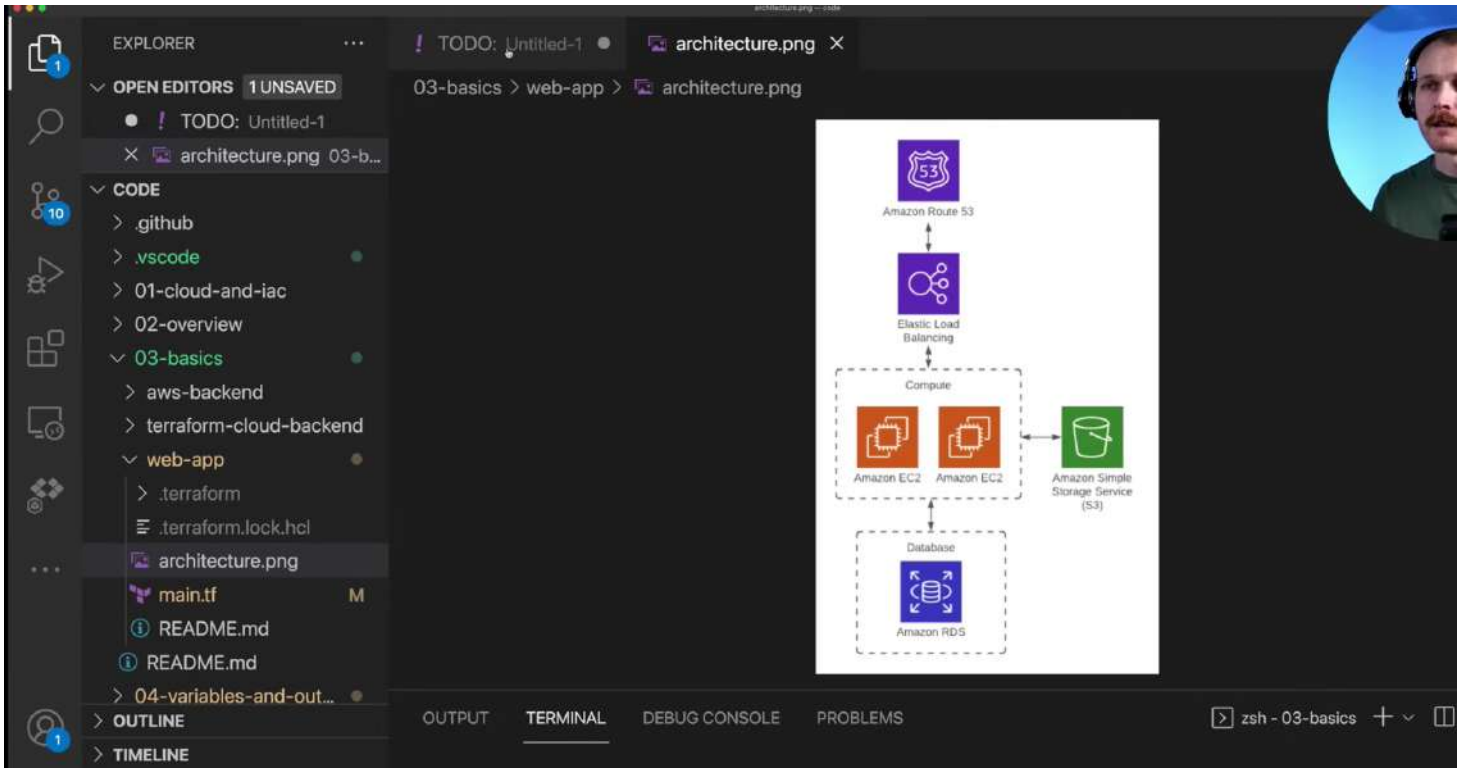
Remote Backend (AWS) Bootstrapping — part 2

```
$ 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)

Remote State File





Variable Types

- Input Variables
  - var.<name>
- Local Variables
  - local.<name>
- Output Variables

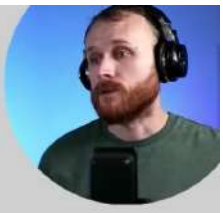
```
variable "instance_type" {
  description = "ec2 instance type"
  type        = string
  default     = "t2.micro"
}

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

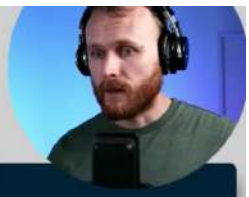
Complex Types:

- 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

HashiCorp Terraform

Expressions + Functions

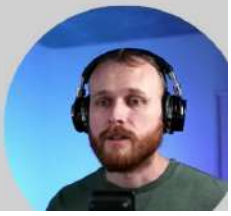
Use the docs!

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

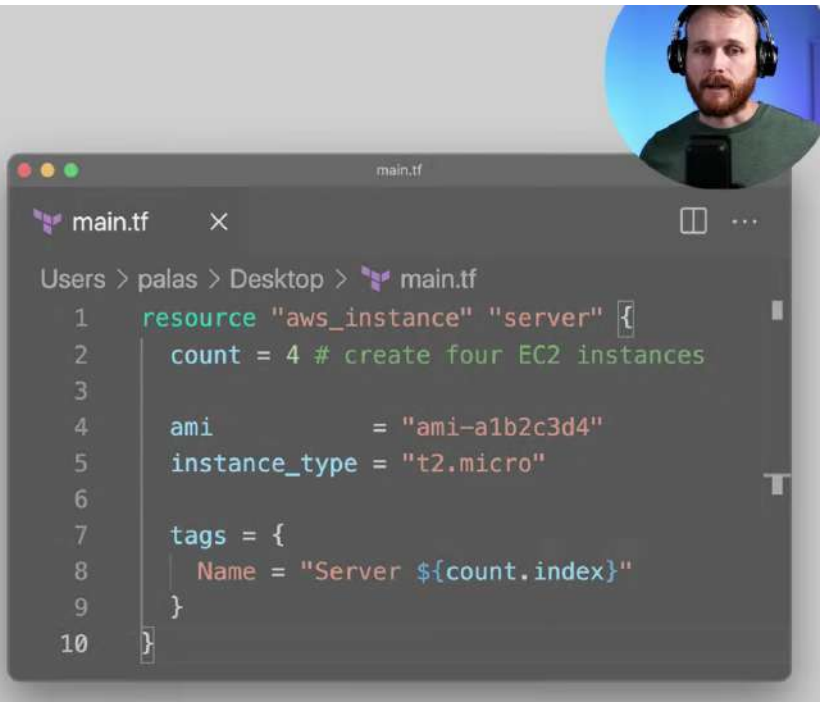
depends_on

- 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*

Meta-Arguments

Count

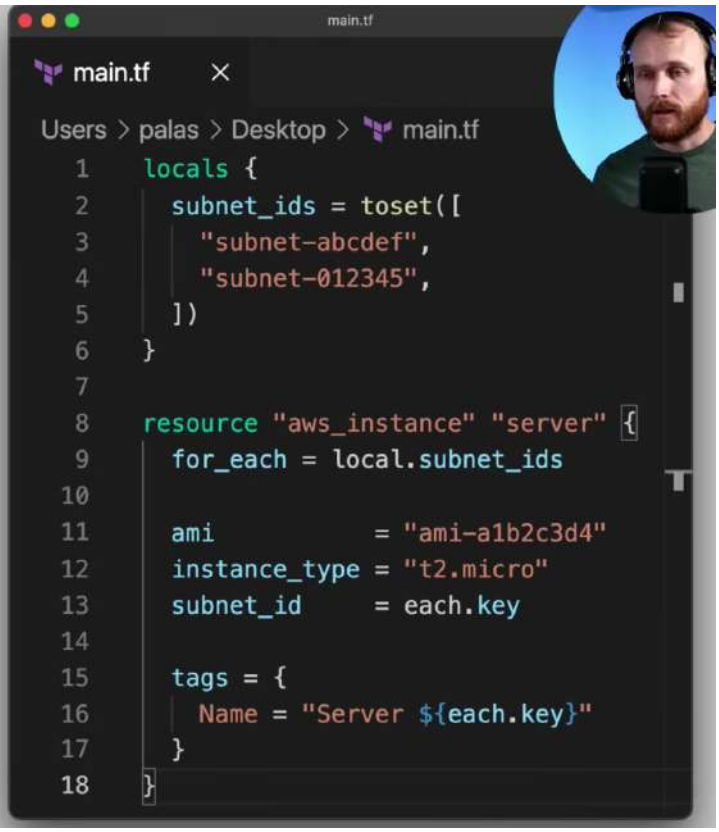
- Allows for creation of multiple resources/modules from a single block
- Useful when the multiple necessary resources are nearly identical



Meta-Arguments

for_each

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



Meta-Arguments

Lifecycle

- A set of meta arguments to control terraform behavior for specific resources
- *create_before_destroy* can help with zero downtime deployments
- i*gnore_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

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

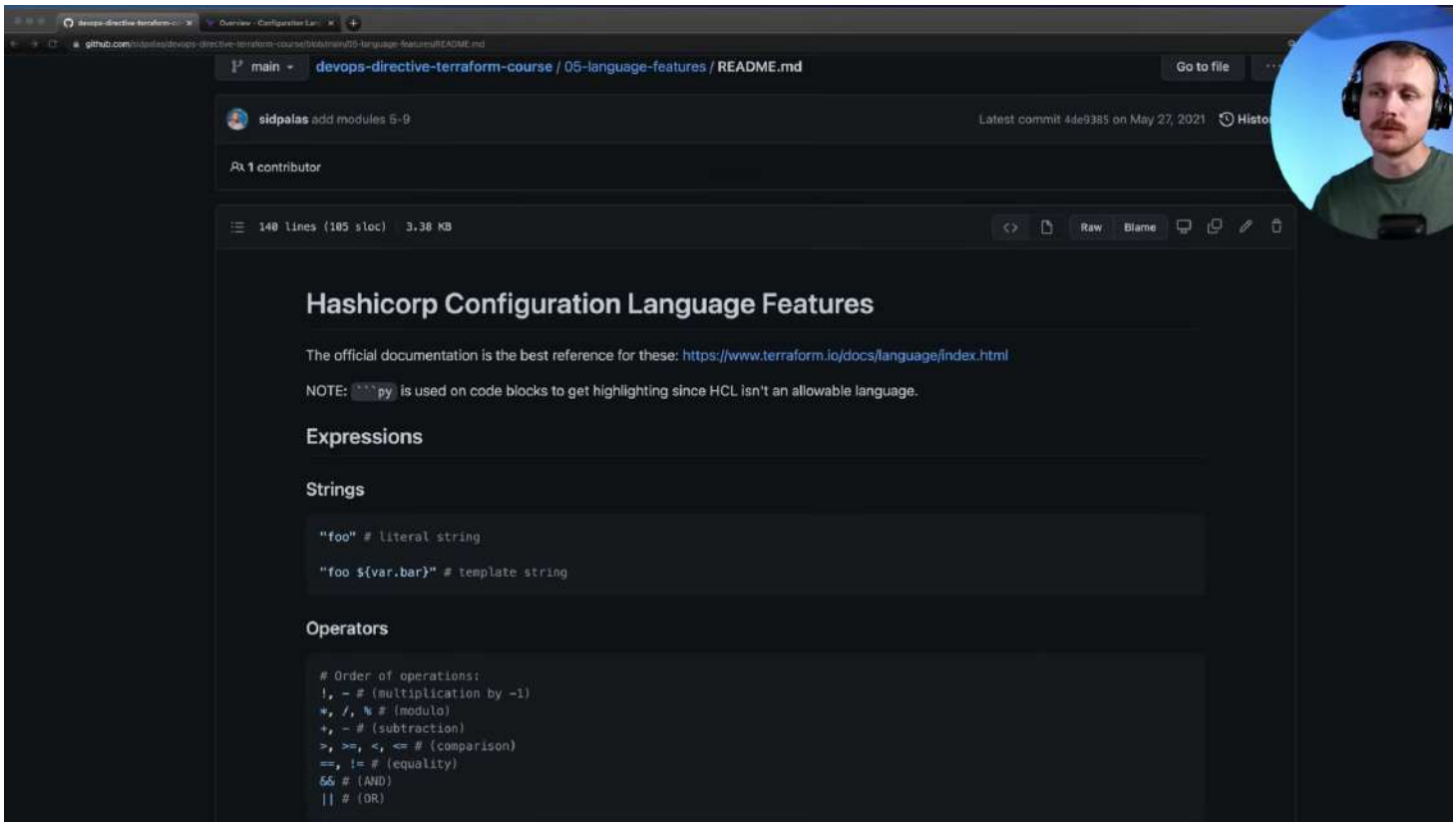
Perform action on local or remote machine

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

Ansible

Chef

Puppet



Part 6

Project Organization + Modules

HashiCorp Terraform

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

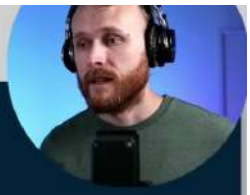
- **Local Path**

```
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
}
```

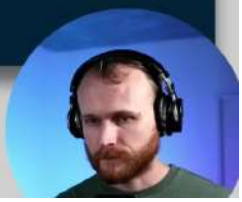
Inputs + Meta-arguments

- Input variables are passed in via module block

Meta-Arguments:

- count
- for_each
- providers
- depends_on

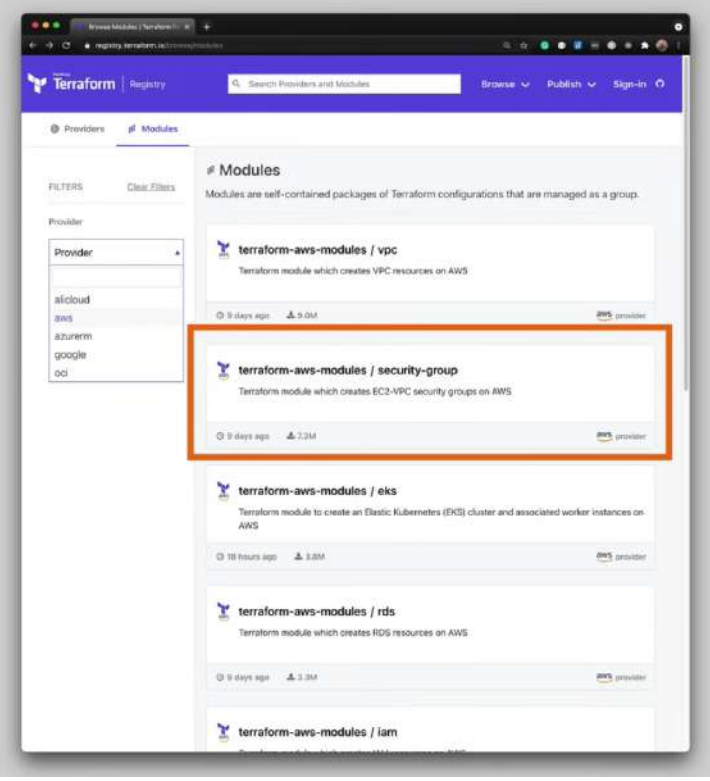
```
module "web_app" {
  source = "../web-app-module"

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

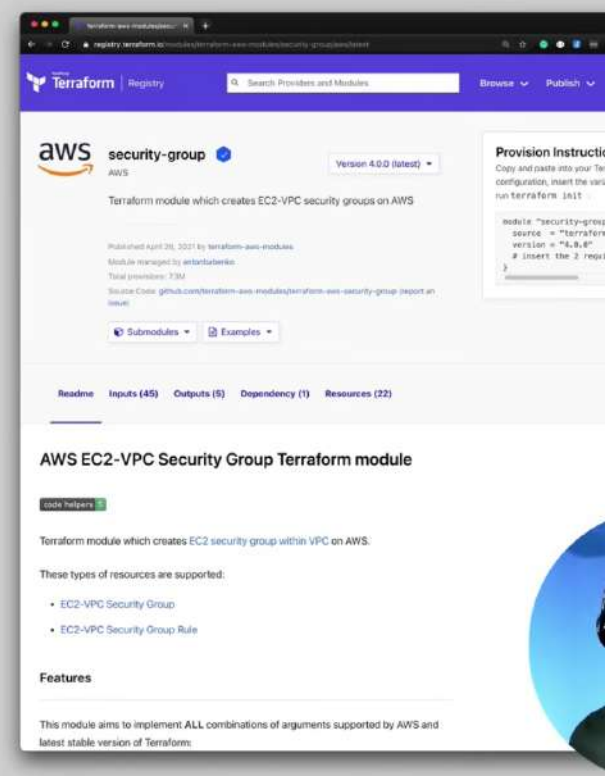


What Makes a Good Module?

- 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



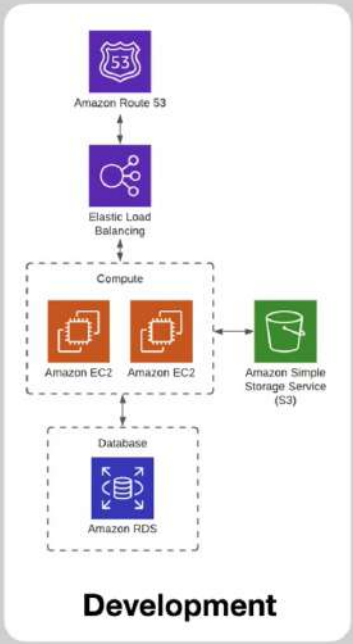


Part 7

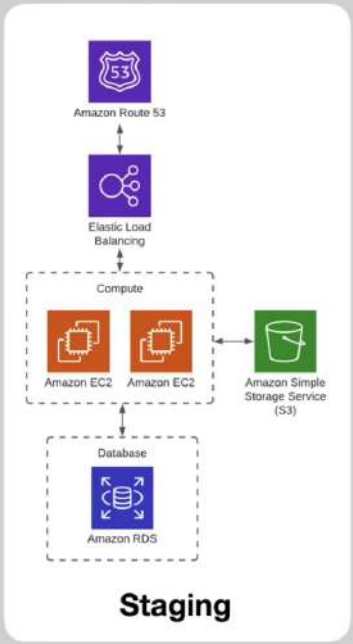
Managing Multiple Environments

HashiCorp Terraform

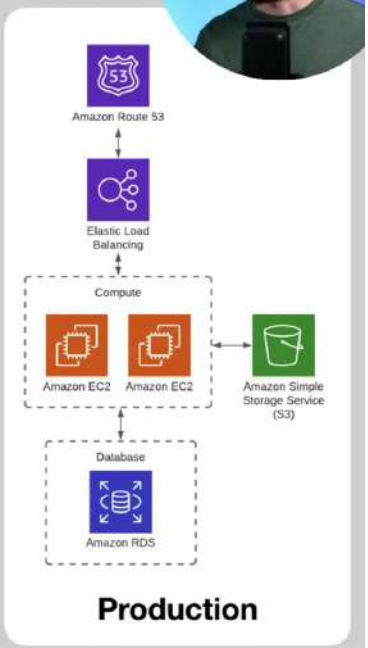
One Config → Multiple Environments



Development



Staging



Production

Two Main Approaches

Workspaces

- Multiple named sections within a single backend

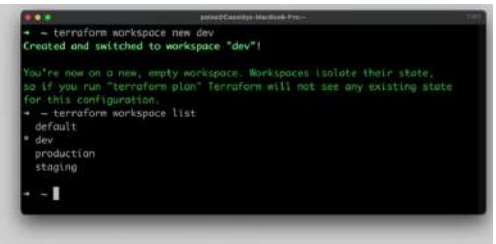
```
→  ~ terraform workspace new dev
Created and switched to workspace "dev"!

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



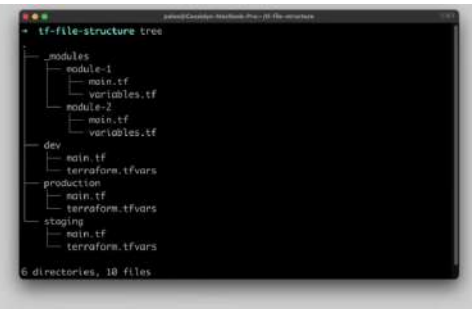
Pros

- 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

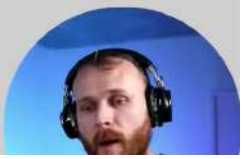


Pros

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

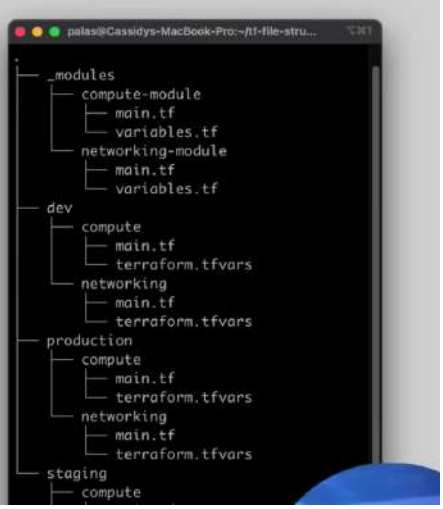
Cons

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



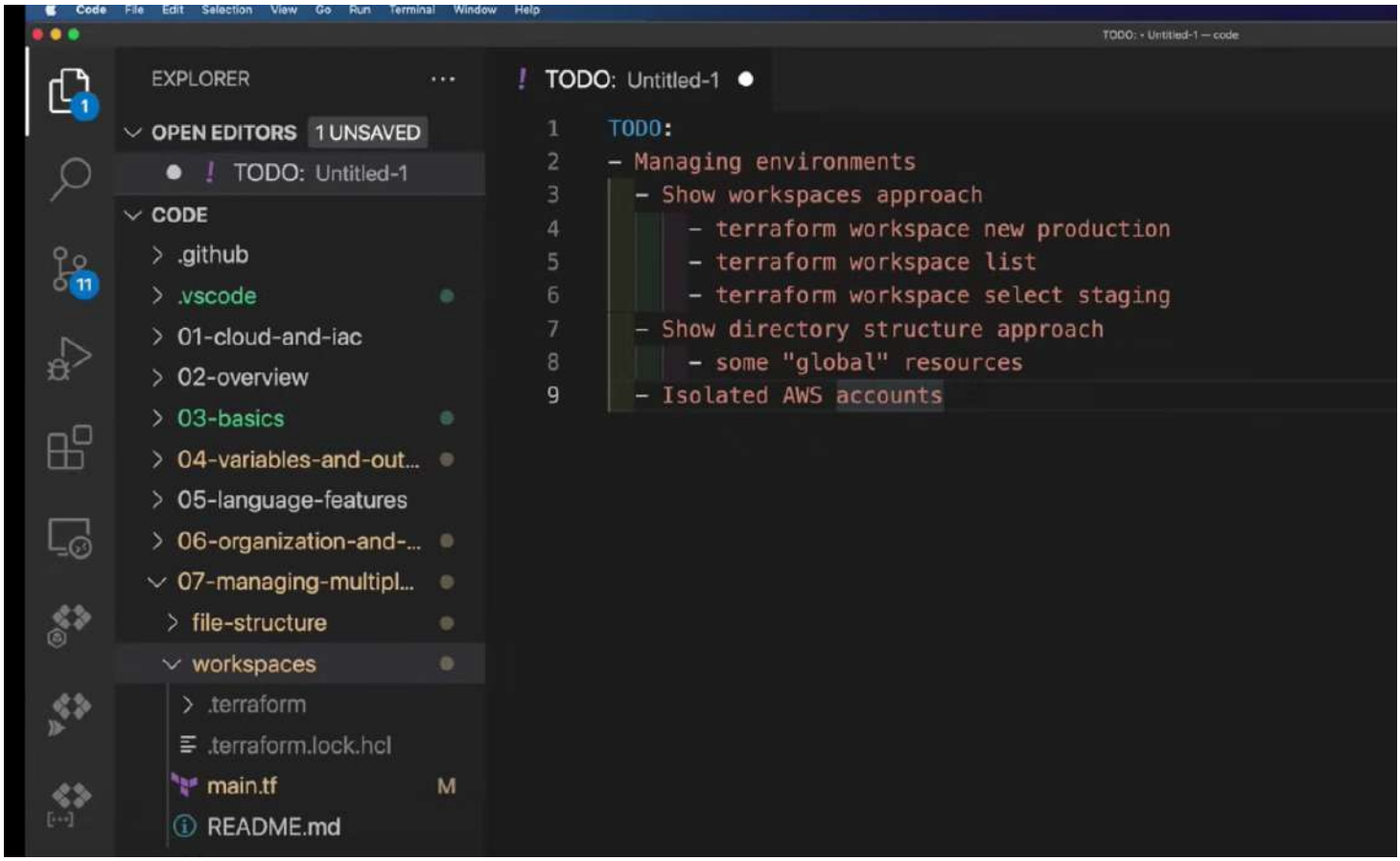
File Structure (environments + components)

- 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



Part 8

Testing Terraform Code

HashiCorp Terraform

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 Sentinal (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 — development (Workspace)

hello_world_test.sh U

development > terraform-course > code > 08-testing > tests >

```
#!/bin/bash
set -euo pipefail

# Change directory to example
cd ../../examples/hello-world

# Create the resources
terraform init
terraform apply -auto-approve

# Wait while the instance boots up
# (Could also use a provisioner in the TF co
sleep 60

# Query the output, extract the IP and make
terraform output -json |\
jq -r '.instance_ip_addr.value' |\
xargs -I {} curl http://{}:8080 -m 10

# If request succeeds, destroy the resources
terraform destroy -auto-approve

```

Automated Testing

Automate the manual test steps…

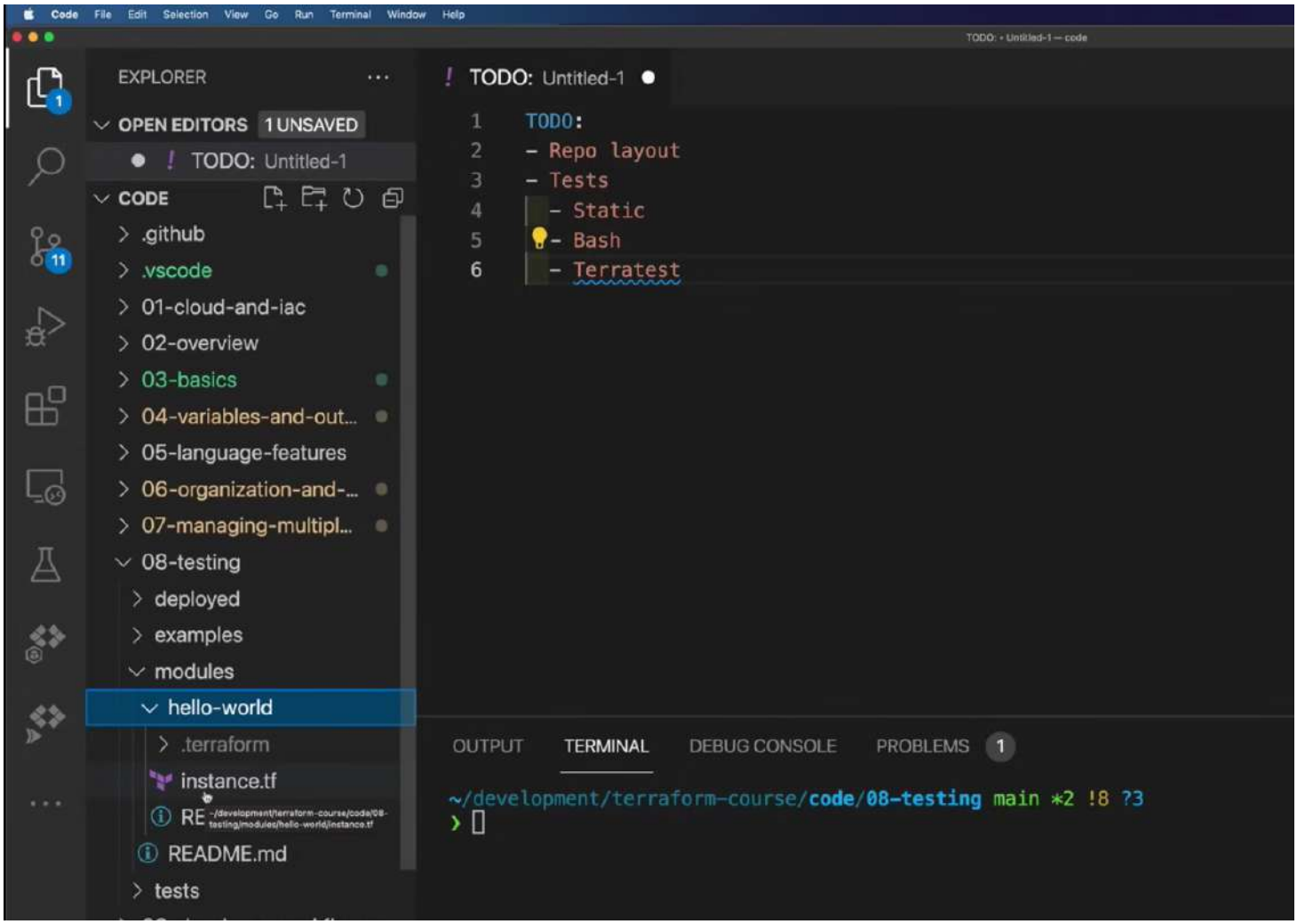
…with Terratest!

hello_world_test.go — development (Workspace)

hello_world_test.go U

development > terraform-course > code > 08-testing > tests > terratest > hell

```
run package tests | run file tests
package test

import (
  "crypto/tls"
  "fmt"
  "testing"
  "time"

  "github.com/gruntwork-io/terratest/modules/http-helper"
  "github.com/gruntwork-io/terratest/modules/terraform"
)

run test | debug test
func TestTerraformHelloWorldExample(t *testing.T) {
  // retryable errors in terraform testing.
  terraformOptions := terraform.WithDefaultRetryableErrors(t, &terra
    TerraformDir: "../../examples/hello-world",
  })

  defer terraform.Destroy(t, terraformOptions)

  terraform.InitAndApply(t, terraformOptions)

  instanceURL := terraform.Output(t, terraformOptions, "url")
  tlsConfig := tls.Config{}
  maxRetries := 30
  timeBetweenRetries := 5 * time.Second

  http_helper.HttpGetWithRetryWithCustomValidation(
    t, instanceURL, &tlsConfig, maxRetries, timeBetweenRetries, valid
  )

}

func validate(status int, body string) bool {
  fmt.Println(body)
  return status == 200
}
```


Code File Edit Selection View Go Run Terminal Window Help
TODO: • Untitled-1 — code
EXPLORER
OPEN EDITORS 1 UNSAVED
TODO: Untitled-1
CODE
.github
.vscode
01-cloud-and-iac
02-overview
03-basics
04-variables-and-out...
05-language-features
06-organization-and-...
07-managing-multipl...
08-testing
deployed
examples
modules
hello-world
.terraform
instance.tf
README.md
tests
TODO: Untitled-1
1 TODO:
2 - Repo layout
3 - Tests
4 - Static
5 - Bash
6 - Terratest
OUTPUT
TERMINAL
DEBUG CONSOLE
PROBLEMS 1
~/development/terraform-course/code/08-testing main *2 !8 ?3