Course Introduction

Jesse Hoch
Training Architect

A CLOUD GURU

In this course

- ✔ Introduction to Terraform
- ✔ Setting Up Your Environment
- ✔ Terraform Fundamentals
- ✔ Terraform Cloud
- ✔ Using Terraform in a CI/CD Environment
- ✔ Terraform and AWS
- ✔ Terraform and Azure
- ✔ Terraform and Kubernetes
- ✔ Troubleshooting

Jesse Hoch
Training Architect

Terraform Fundamentals

- ✔ Terraform CLI
- ✔ Configuration Language
- ✔ Working with Resources
- ✔ Input Variables
- ✔ Declaring Output Variables
- ✔ Declaring Local Variables
- ✔ Modules
- ✔ Module Sources
- ✔ Using Expressions and Functions
- ✔ Backend Configuration
- ✔ Working with States
- ✔ Managing Workspaces

Jesse Hoch
Training Architect

Terraform Cloud

- ✔ Migrating Local Terraform
- ✔ Integrations
- ✔ Workspaces in Terraform Cloud
- ✔ Runs in Terraform Cloud
- ✔ Run States and Stages
- ✔ Users, Teams, and Organizations

Jesse Hoch
Training Architect

Using Terraform in a CI/CD Environment

- ✔ Using GitHub Actions to Automate Terraform
- ✔ Terraform Cloud Setup
- ✔ GitHub Repository Setup
- ✔ Actions Workflow
- ✔ Creating and Merging Pull Requests
- ✔ Verifying a Provisioned Instance

Jesse Hoch
Training Architect

Using Terraform to Create and Manage Infrastructure on AWS

- ✔ Authentication
- ✔ Building Your Infrastructure
- ✔ Making Changes and Destroying Your Infrastructure
- ✔ Defining Input Variables
- ✔ Using Output Variables to Query Data
- ✔ Storing Remote States

Jesse Hoch
Training Architect

A CLOUD GURU

Using Terraform to Create and Manage Infrastructure on Azure

- ✔ Authentication
- ✔ Building Your Infrastructure
- ✔ Making Changes and Destroying Your Infrastructure
- ✔ Defining Input Variables
- ✔ Using Output Variables to Query Data
- ✔ Storing Remote States

Jesse Hoch
Training Architect

A CLOUD GURU

Using Terraform to Create and Manage Infrastructure on Kubernetes

- ✔ Provisioning a Local Kubernetes Cluster
- ✔ Managing Kubernetes Cluster with Terraform
- ✔ Provisioning an EKS Cluster
- ✔ Managing an EKS Cluster with Terraform

Jesse Hoch
Training Architect

A CLOUD GURU

Additionally

- ✔ Troubleshooting
- ✔ Next Steps
- ✔ Hands-on Labs!

Jesse Hoch
Training Architect

A CLOUD GURU

About Me

- ✔ 15+ years of IT experience across multiple roles.
- ✔ Lover of comic books, football, soccer, and a Horror and Sci-Fi fan.
- ✔ Spending time with my family and playing video and board games with my kids.

@JesseHoch1 @Jesse Hoch Jesse Hoch

A CLOUD GURU

Introduction to Terraform

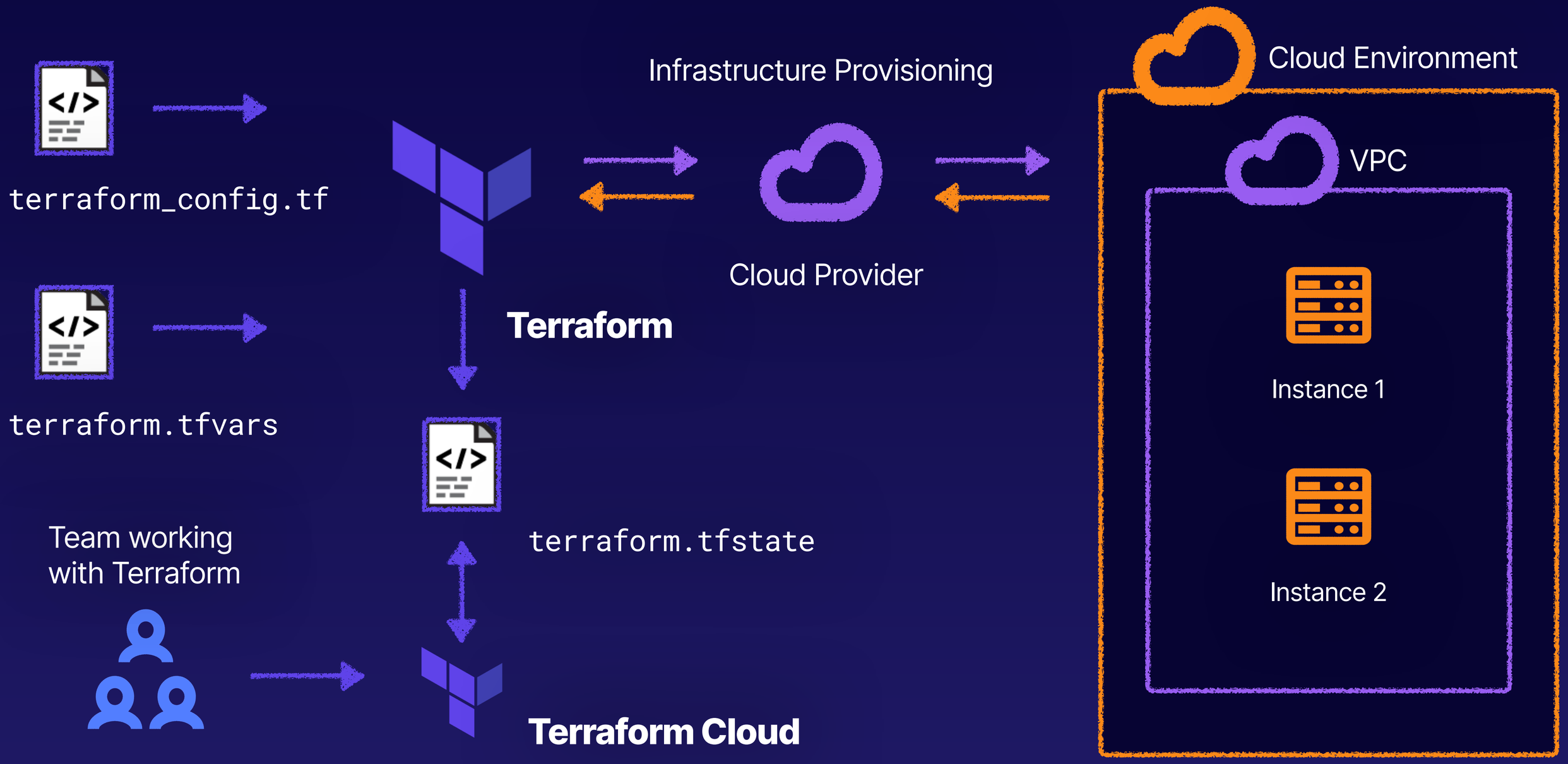
Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

What is Terraform?

It is a tool for building, changing, and versioning infrastructure safely and efficiently, locally or in the cloud. You can manage existing and popular service providers as well as custom in-house solutions.

Terraform Architecture



Key Features

1 Infrastructure as Code

Your infrastructure is described using a high-level configuration syntax. This allows your infrastructure to be versioned and treated as you would any other code. It can also be shared and re-used.

2 Execution Plans

Terraform generates an execution plan with its “planning” step. This shows what Terraform will do when you apply the configuration. This allows you to avoid any surprises when Terraform manipulates infrastructure.

3 Resource Graph

Terraform builds infrastructure as efficiently as possible, and operators get insight into dependencies in their infrastructure. It accomplishes this by building a graph of all your resources, and it haunts the creation and modification of any non-dependent resources.

4 Change Automation

Complex changes can be applied to your infrastructure with minimal interaction. With the combination of the execution plan and resource graph, you will know exactly what Terraform will change and in what order. This will help avoid many possible human errors.

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Introduction to Terraform

A CLOUD GURU

Demo: Installation Methods

Jesse Hoch
TRAINING ARCHITECT

INSTALLATION METHODS

- Manual Installation
- macOS
- Windows
- Linux

A CLOUD GURU

Prerequisites for the Linux Installation Demo:

- Distribution of choice

A CLOUD GURU

Demo Linux Installation Objectives:

- Install the necessary packages
- Add the HashiCorp GPG key
- Add the HashiCorp Linux repo
- Update and install Terraform CLI
- Verify the installation
- Enable tab completion

A CLOUD GURU

Prerequisites for the Windows Installation Demo:

- Chocolatey package manager

A CLOUD GURU

Windows Installation Demo Objectives:

- Download and Install Chocolatey
- Install Terraform
- Verify the installation

A CLOUD GURU

Prerequisites for the macOS Installation Demo:

- Homebrew package manager

A CLOUD GURU

macOS Installation Demo Objectives:

- Download Homebrew
- Update Homebrew and install Terraform
- Verify the installation
- Enable tab completion

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Installation Methods

A CLOUD GURU

Demo: Using Cloud Playground

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Using Cloud Playground

A CLOUD GURU

Demo: Setting Up Docker

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Prerequisites for the Demo:

- Install Terraform CLI
- Install Docker Desktop for Mac or Windows
- Have the `main_docker.tf` file

A CLOUD GURU

Demo Objectives:

- Download and install Docker Desktop
- Start Docker
- Create the `demo-terraform-docker` directory and move into the directory
- Download the `main_docker.tf` file in the working directory and rename it `main.tf`
- Initialize and apply the configuration
- Confirm the deployment and clean up the workspace

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

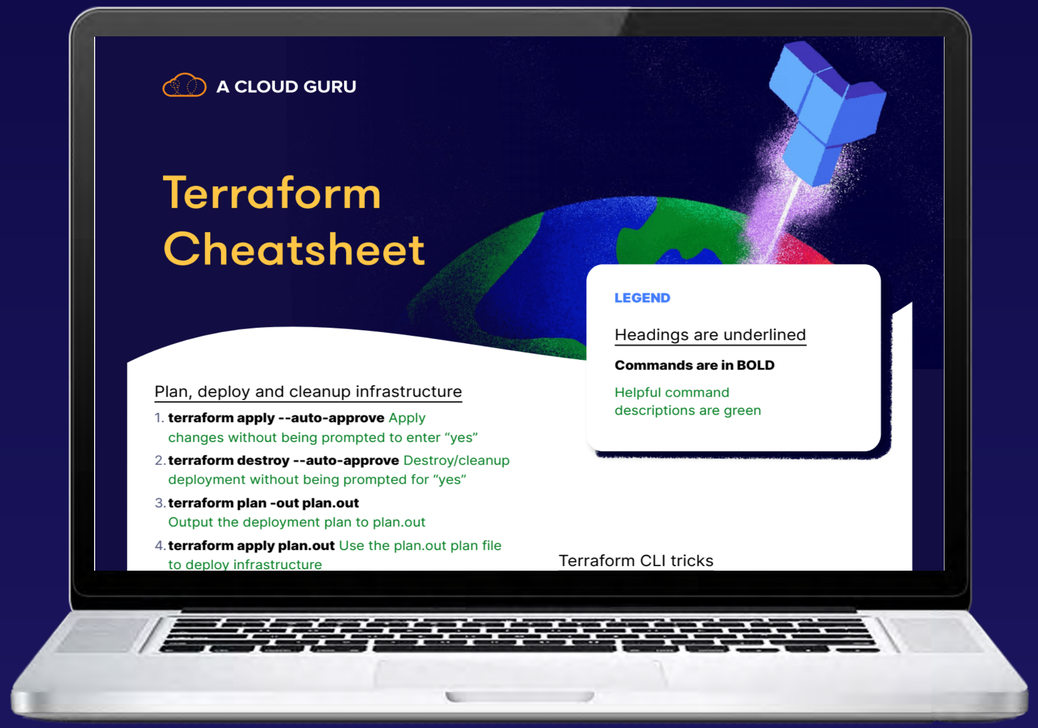
Demo: Setting Up Docker

A CLOUD GURU

Terraform CLI

Jesse Hoch
TRAINING ARCHITECT

Getting Started



Terraform's command line interface (CLI) is accessible via the `terraform` command. This command accepts a variety of subcommands.

The `terraform` command by itself allows you to list the subcommands available.

The `terraform -install-autocomplete` command offers `bash` tab completion-like capabilities.

Basic Commands

A CLOUD GURU

```
$ terraform version
```

Find the Terraform version

```
$ terraform -chdir=<path_to/tf> <subcommand>
```

Switch the working directory

```
$ terraform init
```

Initialize the directory

```
$ terraform plan
```

Create an execution plan

```
$ terraform apply
```

Apply changes

```
$ terraform destroy
```

Destroy the managed infrastructure

Plan, Deploy, and Cleanup Commands

A CLOUD GURU

```
$ terraform plan -out <plan_name>
```

Output a deployment plan

```
$ terraform plan -destroy
```

Output a destroy plan

```
$ terraform apply <plan_name>
```

Apply a specific plan

```
$ terraform apply -target=<resource_name>
```

Only apply changes to a targeted resource

```
$ terraform apply -var my_variable=<variable>
```

Pass a variable via the command line

```
$ terraform providers
```

Get provider info used in configuration

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Terraform CLI

A CLOUD GURU

Configuration Language

Jesse Hoch
TRAINING ARCHITECT

Terraform Language


```
resource "aws_vpc" "main" {
  cidr_block = var.base_cidr_block
}

<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>" {
  # Block body
  <IDENTIFIER> = <EXPRESSION> # Argument
}
```

erraform language's main purpose is to declare resources. This represents infrastructure objects. All the different features are present to accommodate more flexible and convenient resource definition.

Syntax Consists Of:

- *Blocks* are containers for objects like resources.
- *Arguments* assign a value to a name.
- *Expressions* represent a value.

Directories and Files

File Extension

Plain text files with the *.tf* file extension are where code in Terraform language is stored. You can also use a json variant of the language which uses the *.tf.json* file extension.

Directories and Modules

Modules are a collection of *.tf* and/or *.tf.json* files kept together in a directory. A module consists of only the top-level config files in the directory. A nested directory is treated as a separate module and may not be automatically included.

Text Encoding

These plain text files must use UTF-8 encoding and usually use Unix-style line endings (LF), though it can also use Windows-style line endings (CRLF) as well.

Root Module

Terraform configuration consists of a root module and a few child modules. The root module is the working directory where Terraform is invoked.

Override Files


```
resource "aws_instance" "web" {
  instance_type = "t2.micro"
  ami           = "ami-408c7f28"
}
```

Terraform config example

```
resource "aws_instance" "web" {
  ami = "foo"
}
```

override.tf file

```
resource "aws_instance" "web" {
  instance_type = "t2.micro"
  ami           = "foo"
}
```

How the merge is treated

Config Syntax


```
image_id = “terra123"
```

Argument example

```
resource "aws_instance" "example" {
  ami = "abc123"

  network_interface {
    # ...
  }
}
```

Block example

Identifiers

Identifiers are things like argument names, block type names, resources, input variables, and etc.

Comments

- # begins a single-line comment, ending at the of a line.
- // also begins a single-line comment, as an alternative to #.
- /* and */ are start and end delimiters for a comment that might span over multiple lines.

JSON Config Syntax


```
{
  "variable": {
    "example": {
      "default": "hello"
    }
  }
}
```

Variable example

Equivalent native syntax

```
variable "example" {
  default = "hello"
}

resource "aws_instance" "example" {
  instance_type = "t2.micro"
  ami           = "ami-abc123"
}
```

```
{
  "resource": {
    "aws_instance": {
      "example": {
        "instance_type": "t2.micro",
        "ami": "ami-abc123"
      }
    }
  }
```

Instance type block example

JSON Config vs Terraform Config

A CLOUD GURU

```
{
  "resource": {
    "aws_instance": {
      "example": {
        "lifecycle": {
          "create_before_destroy": true
        }
      }
    }
  }
}
```

JSON config example

```
resource "aws_instance" "example" {
  lifecycle {
    create_before_destroy = true
  }
}
```

Terraform config equivalent

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Configuration Language

A CLOUD GURU

Working With Resources

Jesse Hoch
TRAINING ARCHITECT

Resources


```
resource "aws_instance" "web" {
  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"
}
```

Resources are the most important part of the Terraform language. Resource blocks describe infrastructure objects like virtual networks, compute instances, or components like DNS records.

Resource Types:

- ***Providers***, which are plugins for Terraform that offers a collection of resources types.
- ***Arguments***, which are specific to the selected resource type.
- ***Documentation***, which every provider uses to describe its resource types and arguments.

Meta-Arguments

A CLOUD GURU

depends_on

Specify hidden dependencies.

provider

Select a non-default provider configuration.

count

Create multiple resource instances according to a count.

lifecycle

Set lifecycle customizations.

for_each

Create multiple instances according to a map or a set of strings.

provisioner and connection

Take extra actions after resource creation.

Operation Timeouts


```
resource "aws_db_instance" "example" {
  # ...

  timeouts {
    create = "60m"
    delete = "2h"
  }
}
```

There are some resource types that provide special *timeouts*, nested block arguments that allow for customization of how long certain operations are allowed to take before they are deemed failed.

timeouts string examples

- "60m"
- "10s"
- "2h"

How Configuration Is Applied

Create

Create resources that exist in the configuration but are not associated with a real infrastructure object in the state.

Update in-place

Update in-place resources whose arguments have changed.

Destroy

Destroy resources that exist in the state but no longer exist in the configuration.

Destroy and re-create

Destroy and re-create resources whose arguments have changed, but which cannot be updated in-place due to remote API limitations.

Resource Behavior

Accessing Resource Attributes

- Expressions within Terraform modules
- Read-only attributes with information obtained from remote APIs
- Provider-included data sources

Local-only Resources

Specialized resource types that operate only within Terraform itself, calculating some results and saving those results in the state for future use. Examples: ssh keys, self-signed certs, and random ids.

Resource Dependencies

Most resource dependencies are handled automatically. Terraform analyzes any expressions within a resource block to find references to other objects, and treats those references as ordering requirements when creating, updating, or destroying resources. It's usually not necessary to manually specify dependencies between resources, however some dependencies cannot be recognized implicitly in the configuration.

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Working With Resources

A CLOUD GURU

Input Variables

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Input Variables Are Like Function Arguments

Input variables serve as parameters for a Terraform module. They allow aspects of the module to be customized without altering the actual module. This allows modules to be shared between different configurations.

Declaring an Input Variable


```
variable "image_id" {
  type = string
}

variable "availability_zone_names" {
  type    = list(string)
  default = ["us-west-1a"]
}

variable "docker_ports" {
  type = list(object({
    internal = number
    external = number
    protocol = string
  }))
  default = [
    {
      internal = 8300
      external = 8300
      protocol = "tcp"
    }
  ]
}
```

Variable Block Example

The name of a variable can be any valid ***identifier*** except for `source`, `version`, `providers`, `count`, `for_each`, `lifecycle`, `depends_on`, and `locals`.

Arguments and Constraints

A CLOUD GURU

Optional Arguments for Variable Declaration

- default
- type
- description
- validation
- sensitive

Type Constraints

- string
- number
- bool

Type Constructors

- list(<type>)
- set(<type>)
- map(<type>)
- object({<attribute> = <type>, …})
- tuple([<type>, …])

Argument Examples

A CLOUD GURU

Input Variable Description

```
variable "image_id" {
  type        = string
  description = "The id of the machine image (AMI) to use for the server."
}
```

Custom Validation Rules

```
variable "image_id" {
  type        = string
  description = "The id of the machine image (AMI) to use for the server."

  validation {
    condition     = length(var.image_id) > 4 && substr(var.image_id, 0, 4) == "ami-"
    error_message = "The image_id value must be a valid AMI id, starting with \"ami-\"."
  }
}
```

Argument Examples

A CLOUD GURU

Using the Sensitive Argument

```
variable "user_information" {
  type = object({
    name    = string
    address = string
  })
  sensitive = true
}

resource "some_resource" "a" {
  name    = var.user_information.name
  address = var.user_information.address
}
```

Resulting Terraform Output

```
Terraform will perform the following
actions:

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

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

Using Input Variable Values

A CLOUD GURU

A value can be accessed from an expression, such as `var.<var_name>`.

```
resource "aws_instance" "example" {
  instance_type = "t2.micro"
  ami           = var.image_id
}
```

A value assigned to a variable can only be accessed in an expression within the module it was declared.

How to Assign Values to Root Modules Variables

A CLOUD GURU

How to assign values:

- In a Terraform Cloud workspace
- Individually, with the `-var` command line option
- In variable definitions files like `.tfvrs` or `.tfvars.json`
- As environment variables

Variables on the command line:

```
$ terraform apply -var="image_id=ami-abc123"
$ terraform apply -var='image_id_list=["ami-abc123","ami-def456"]' -var="instance_type=t2.micro"
$ terraform apply -var='image_id_map={"us-east-1":"ami-abc123","us-east-2":"ami-def456"}'
```

How to Assign Values to Root Modules Variables

A CLOUD GURU

Calling the variable definition file from the command line:

```
$ terraform apply -var-file="testing.tfvars"
```

Automatically loaded definition files:

- Files named exactly `terraform.tfvars` or `terraform.tfvars.json`.
- Any files with names ending in `.auto.tfvars` or `.auto.tfvars.json`.

Variable definition file example:

```
image_id = "ami-abc123"
availability_zone_names = [
  "us-east-1a",
  "us-west-1c",
]
```

Environment Variables

Terraform searches the environment for environment variables named `TF_VAR_` followed by the name of the variable.

```
$ export TF_VAR_image_id=ami-abc123
$ terraform plan
...
```

Terraform matches the variable name exactly as given in configuration on operating systems where the environment variable is case-sensitive. The required environment variable name usually includes a mix of upper and lowercase letters, like in the example above.

Precedence

The order in which variables are loaded:

- `Environment variables`
- `terraform.tfvars`
- `terraform.tfvars.json`
- `*.auto.tfvars or *.auto.tfvars.json`
- `Any command-line options like -var and -var-file`

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Input Variables

A CLOUD GURU

Declaring Output Variables

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Output Variables Are Like Return Values

They have many uses:

- ✓ A child module can use them to expose a subset of resource attributes to the parent module.
- ✓ A root module can use them to print values in the CLI.
- ✓ Root module outputs can be accessed by other configurations via the `terraform_remote_state` data source.

Declaring an Output Variable

A CLOUD GURU

Output Block Example

```
output "instance_ip_addr" {
  value = aws_instance.server.private_ip
}
```

The `name` label after the `output` keyword must be a valid identifier.

The `value` argument takes an expression whose result will be returned to the user.

Arguments and Constraints

When accessing child module outputs in a parent module, the outputs of the child modules are available in expressions as `module.<module_name>.<output_name>`.

Optional Arguments for Variable Declaration

- `description`
- `sensitive`
- `depends_on`

Argument Examples

A CLOUD GURU

Using the `description` Output Argument

```
output "instance_ip_addr" {
  value       = aws_instance.server.private_ip
  description = "The private IP address of the main server instance."
}
```

The `description` should clearly explain the purpose of the `output` and what kind of `value` is expected.

Argument Examples

A CLOUD GURU

Using the sensitive Argument

```
# main.tf

module "foo" {
  source = "./mod"
}

resource "test_instance" "x" {
  some_attribute = module.mod.a
}

output "out" {
  value     = "xyz"
  sensitive = true
}

output "a" {
  value     = "secret"
  sensitive = true
}
```

Resulting Terraform Output

```
Terraform will perform the following
actions:

  # test_instance.x will be created
  + resource "test_instance" "x" {
      + some_attribute    = (sensitive)
    }

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

Changes to Outputs:
  + out = (sensitive value)
```

Argument Examples

A CLOUD GURU

Using the depends_on Output Argument

```
output "instance_ip_addr" {
  value       = aws_instance.server.private_ip
  description = "The private IP address of the main server instance."

  depends_on = [
    # Security group rule must be created before this IP address could
    # actually be used, otherwise the services will be unreachable.
    aws_security_group_rule.local_access,
  ]
}
```

The depends_on argument should be used as a last resort. You should always include a comment explaining why it is being used.

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Declaring Output Variables

A CLOUD GURU

Declaring Local Variables

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Local Variables Are Like a Function's Temporary Local Variables

Allow you to:

- ✓ Assigns a name to an expression.
- ✓ Allows you to use the variable multiple times within a module without repeating it.

Declaring a Local Variable

A CLOUD GURU

Locals Block Example

```
locals {
  service_name = "forum"
  owner        = "Community Team"
}
```

A set of related **local** values can be declared together in a single block.

The **expressions** are not limited to literal constants; they can reference other values in the module.

Using Local Values

A CLOUD GURU

When a local value is declared, you can reference it in expressions as `local.<name>`.

```
resource "aws_instance" "example" {
  # ...

  tags = local.common_tags
}
```

Local values can only be accessed in expressions within the module where they were declared.

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Declaring Local Variables

A CLOUD GURU

Modules

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Modules Are Containers for Multiple Resources

A module can consist of a collection of `.tf` as well as `.tf.json` files kept together in a directory.

Module Types

3 Module Types

- Root Modules - You need at least one root module.
- Child Modules - Modules that are called by the root module.
- Published Modules - Modules loaded from a private or public registry.

Calling a Child Module

A CLOUD GURU

Module Block

```
module "servers" {
  source = "./app-cluster"

  servers = 5
}
```

A **root module** that includes a module block is calling a **child module**. The label after the `module` keyword is a local name that can be used to refer to the module.

Module Arguments

4 Module Argument Types

- The `source` argument is required for all modules.
- The `version` argument is recommended for modules from a registry.
- The `input variable` arguments.
- The meta-arguments like `for_each` and `depends_on`.

Argument Examples

A CLOUD GURU

Version Argument

```
module "consul" {
  source  = "hashicorp/consul/aws"
  version = "0.0.5"

  servers = 3
}
```

Meta-Arguments

- count
- for_each
- providers
- depends_on

Accessing Module Output Values

A CLOUD GURU

Module Output Values

```
resource "aws_elb" "example" {
  # ...

  instances = module.servers.instance_ids
}
```

Child modules can declare **output values** to selectively export values which are accessible by the **calling module**.

Transferring Resource State

Use the `terraform state mv` command to inform Terraform that the child module block has moved to a different module. When you split code into other child modules, or when moving resource blocks between modules, you can cause Terraform to see the new location of the module block as an entirely different resource.

When passing resource addresses

- Resources within the child modules must be prefixed with `module.<module_name>.`.
- If a module is called with `count` or `for_each`, it must be prefixed with `module.module_name>[<index>].` instead.

Tainting Resources within a Module

The `taint` command can be used to taint specific resources within a module.

```
$ terraform taint module.salt_master.aws_instance.salt_master
```

It is **not** possible to **taint** an entire module. Instead, each resource within the **module** **must** be **tainted** separately.

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Modules

A CLOUD GURU

Modules Sources

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Module sources tell Terraform where to look for source code.

Terraform uses the module source during the module installation step of `terraform init`.

Module Arguments

8 Module Source Types

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

Local Paths

Local Paths Example:

```
module "consul" {
  source = "./consul"
}
```

A **local path** must begin with either `./` or `../` to indicate that it is indeed a **local path**. **Local paths** are not installed in the same sense as other sources.

Terraform Registry

A CLOUD GURU

Public Registry Example:

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

Other Registry Example:

```
module "consul" {
  source = "app.terraform.io/example-corp/k8s-cluster/azurerm"
  version = "1.1.0"
}
```

GitHub

GitHub HTTPS Example:

```
module "consul" {
  source = "github.com/hashicorp/example"
}
```

GitHub SSH Example:

```
module "consul" {
  source = "git@github.com:hashicorp/example.git"
}
```

Bitbucket

Bitbucket Example:

```
module "consul" {
  source = "bitbucket.org/hashicorp/terraform-consul-aws"
}
```

Terraform will also recognize un-prefixed **bitbucket.org** URLs and interpret them as **Bitbucket repositories**. This only works for public repositories.

Generic Git Repository

A CLOUD GURU

Generic GitHub Repo Example:

```
module "vpc" {
  source = "git::https://example.com/vpc.git"
}

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

Selecting a Revision Example:

```
module "vpc" {
  source = "git::https://example.com/vpc.git?ref=v1.2.0"
}
```

Generic Mercurial Repository

Generic Mercurial Repo Example:

```
module "vpc" {
  source = "hg::http://example.com/vpc.hg"
}
```

Selecting a Revision Example:

```
module "vpc" {
  source = "hg::http://example.com/vpc.hg?ref=v1.2.0"
}
```

HTTP URLs

If the GET request is successful, Terraform looks in the following locations:

- The value of a response header field named `X-Terraform-Get`.
- If the response is an HTML page, a meta element with the name `terraform-get`.

Meta Element Example:

```
<meta name="terraform-get"
        content="github.com/hashicorp/example" />
```

Fetching Archives over HTTP

HTTP URLs Example:

```
module "vpc" {
  source = "https://example.com/vpc-module.zip"
}
```

Extensions Terraform Recognizes:

- zip
- tar.bz2 and tbz2
- tar.gz and tgz
- tar.xz and txz

Archive Argument Example:

```
module "vpc" {
  source = "https://example.com/vpc-module?archive=zip"
}
```

S3 Bucket

Locations the module installer looks for AWS creds:

- The `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment.
- The default profile in the `.aws/credentials` file in your home directory.
- If running on an EC2 instance, temporary credentials associated with the instance's IAM instance profile.

S3 Bucket Example:

```
module "consul" {
  source = "s3::https://s3-eu-west-1.amazonaws.com/examplecorp-terraform-modules/vpc.zip"
}
```

GCS Bucket

To Set Credentials:

- Enter the path of your service account key file in the `GOOGLE_APPLICATION_CREDENTIALS` environment variable.
- If you're running Terraform from a GCE instance, the default credentials are automatically available.
- On your computer, you can make your Google identity available by running `gcloud auth application-default login`.

GCS Bucket Example:

```
module "consul" {
  source = "gcs::https://www.googleapis.com/storage/v1/modules/foomodule.zip"
}
```

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Module Sources

A CLOUD GURU

Using Expressions and Functions

Jesse Hoch
TRAINING ARCHITECT

The Difference between Expressions and Functions

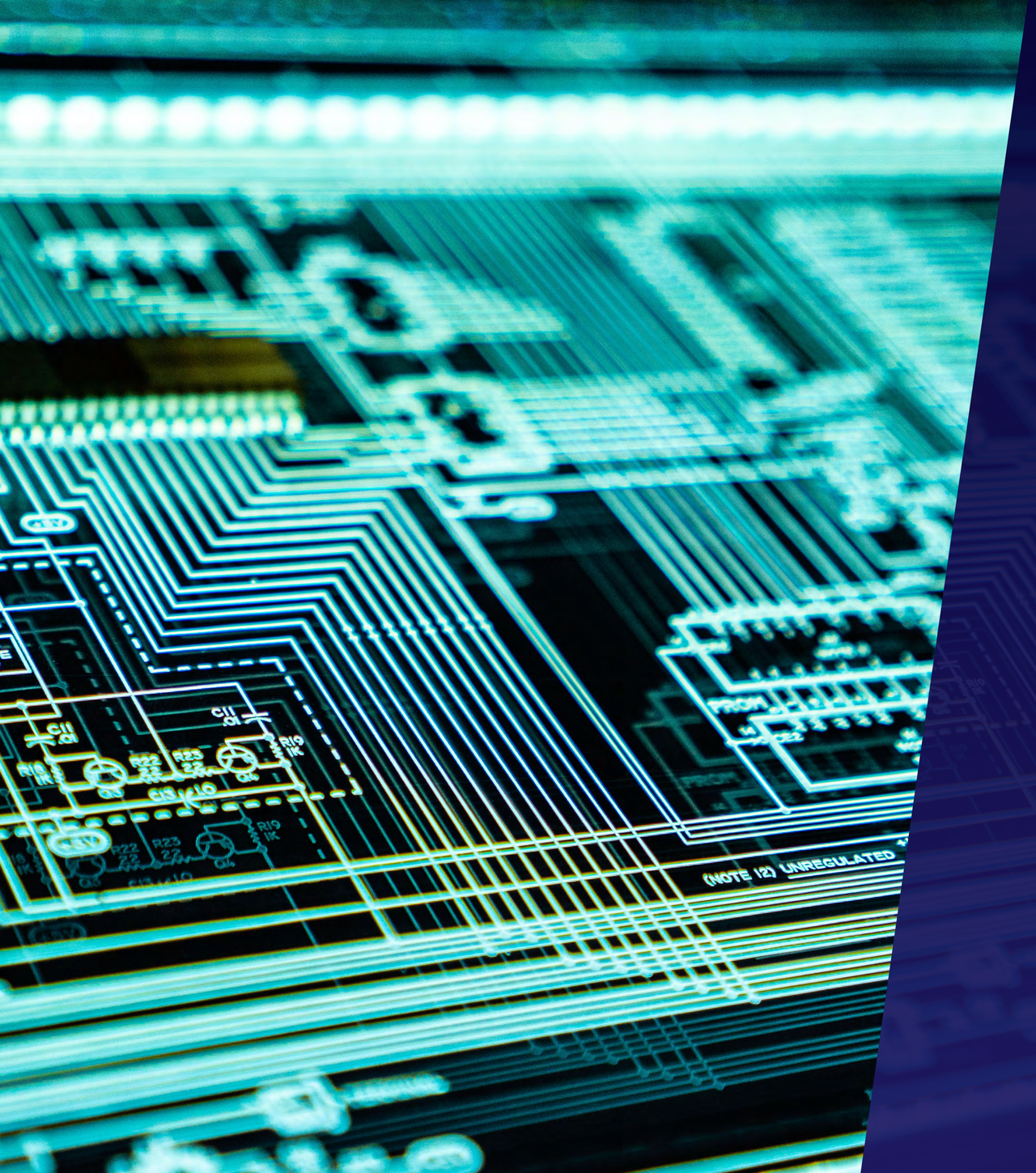
EXPRESSIONS

Expressions are used to reference/ compute values within a configuration.

VS

FUNCTIONS

Functions are used to transform and combine values within expressions.



A CLOUD GURU

EXPRESSIONS

The simplest expressions are literal values, like ACG or 1, but the Terraform language also allows more complex expressions, such as references to data exported by resources and a number of built-in functions.

The Terraform language uses the following type values:

- `string`
- `number`
- `bool`
- `list/tuple`
- `map/object`
- `null`

7 Types of Named Values Available in Terraform

- Resources
- Input Variables
- Local Values
- Child Module Outputs
- Data Sources
- Filesystem and Workspace info
- Block-local Values

A CLOUD GURU

Conditional Expressions

A CLOUD GURU

Conditional Expression Syntax:

```
condition ? true_val : false_val
```

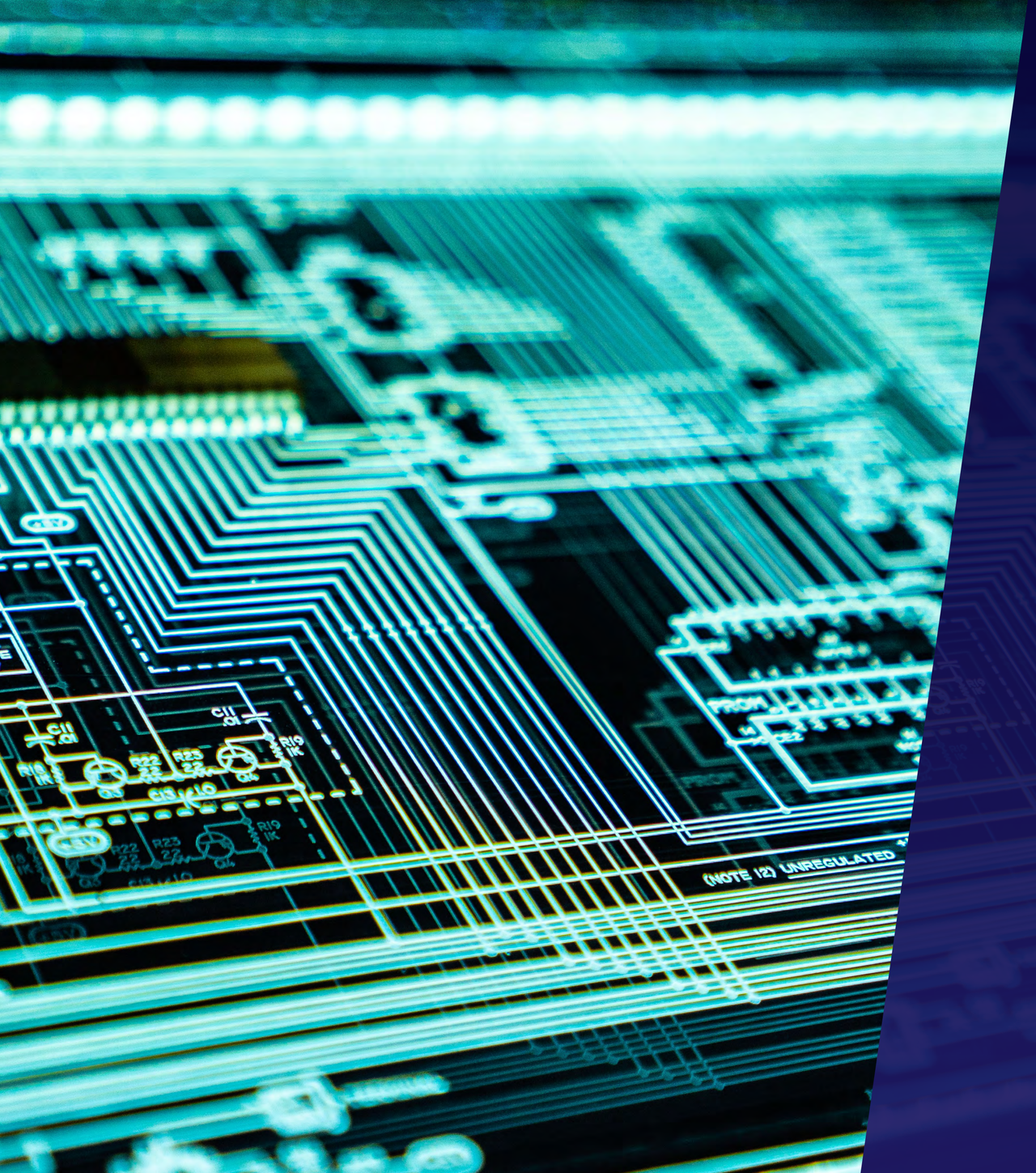
Defining Defaults to Replace Invalid Values:

```
var.a != "" ? var.a : "default-a"
```

Result Type Examples:

```
var.example ? 12 : "hello"
```

```
var.example ? tostring(12) :
```



A CLOUD GURU

FUNCTIONS

The Terraform language includes a number of built-in **functions** you can call from within expressions to transform and combine values.

Function Calls

A CLOUD GURU

Built-In Function Syntax:

```
<FUNCTION NAME>(<ARGUMENT 1>, <ARGUMENT 2>)
```

Function Example:

```
min([55, 2453, 2]...)
```

Using Sensitive Data as a Function Argument:

```
> local.baz
{
  "a" = (sensitive)
  "b" = "dog"
}
> keys(local.baz)
(sensitive)
```

Function Calls

Function Call General Syntax:

```
max(5, 12, 9)
```

Trying Function Calls Using Terraform Expression Console:

```
terraform console
```

```
> max(5, 12, 9)
12
```

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Using Expressions and Functions

A CLOUD GURU

Backend Configuration

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Each Terraform configuration can specify a backend.

- HashiCorp recommends Terraform beginners use the default **local** backend.
- If you are working with a team and managing a large infrastructure, HashiCorp recommends you use a **remote** backend.

Where Backends Are Used

Where They Come From

Terraform includes a built-in selection of backends, and these are the only backends. You cannot load additional backends as plugins.

Where They Are Used

Backend configuration is only used by Terraform CLI. Terraform Cloud and Enterprise always use their own state storage.

What They Do

Two areas of Terraform's behavior are determined by the backend:

- Where state is stored
- Where operations are performed

Backend Block

Backend Block Example:

```
terraform {
  backend "remote" {
    organization = “corp_example"

    workspaces {
      name = “ex-app-prod"
    }
  }
}
```

Backend Block Limitations:

- A configuration can only provide one backend block.
- A backend block cannot refer to named values.

Three Things to Know:

- When the backend changes in a configuration, you must run `terraform init`.
- When the backend changes, Terraform gives you the option to migrate your state.
- HashiCorp recommends you manually backup your state. Simply copy the `terraform.tfstate` file.

Enhanced Backend: local

A CLOUD GURU

local Example Configuration:

```
terraform {
  backend "local" {
    path = "/path/to/terraform.tfstate"
  }
}
```

Configuration Variables:

- `path` - The path to the `tfstate` file.
- `workspace_dir` - The path to non-default workspaces.

Data Source Configuration Example:

```
data "terraform_remote_state" “zland” {
  backend = "local"

  config = {
    path = "${path.module}/../../terraform.tfstate"
  }
```

Enhanced Backend: remote

A CLOUD GURU

Remote Config Block with Workspace:

```
terraform {
  backend "remote" {
    hostname = "app.terraform.io"
    organization = "company"

    workspaces {
      name = "my-app-prod"
    }
  }
}
```

Using the CLI Input:

```
# main.tf
terraform {
  required_version = "~> 0.12.0"

  backend "remote" {}
}
```

Backend Config:

```
# backend.hcl
workspaces { name = "workspace" }
hostname     = "app.terraform.io"
organization = "company"
```

Using Backend File with `terraform init`:

```
$ terraform init -backend-config=backend.hcl
```

Enhanced Backend: remote

A CLOUD GURU

Data Source Configuration:

```
data "terraform_remote_state" "foo" {
  backend = "remote"

  config = {
    organization = "company"

    workspaces = {
      name = "workspace"
    }
  }
}
```

Configuration Variables:

- `hostname` (Optional)
- `organization` (Required)
- `token` (Optional)
- `workspaces` (Required)
  - `name` (Optional)
  - `prefix` (Optional)

Standard Backend: S3

A CLOUD GURU

Example Block Configuration:

```
terraform {
  backend "s3" {
    bucket = "mybucket"
    key    = "path/to/my/key"
    region = "us-east-1"
  }
}
```

Data Source Configuration:

```
data "terraform_remote_state" "network" {
  backend = "s3"
  config = {
    bucket = "terraform-state-prod"
    key    = "network/terraform.tfstate"
    region = "us-east-1"
  }
}
```

S3 Bucket Permissions:

- s3:ListBucket
- s3:GetObject
- s3:PutObject

DynamoDB Table Permissions:

- dynamodb:GetItem
- dynamodb:PutItem
- dynamodb:DeleteItem

Standard Backend: azurerm

Using Azure CLI or a Service Principal:

```
terraform {
  backend "azurerm" {
    resource_group_name  = "StorageAccount-ResourceGroup"
    storage_account_name = "abcd1234"
    container_name       = "tfstate"
    key                  = "prod.terraform.tfstate"
  }
}
```

Using Azure AD:

```
terraform {
  backend "azurerm" {
    storage_account_name = "abcd1234"
    container_name       = "tfstate"
    key                  = "prod.terraform.tfstate"
    use_azuread_auth     = true
    subscription_id      = "00000000-0000-0000-0000-000000000000"
    tenant_id            = "00000000-0000-0000-0000-000000000000"
  }
}
```

Standard Backend: azurerm

Data Source Config Using a Service Principal:

```
data "terraform_remote_state" "foo" {
  backend = "azurerm"
  config = {
    storage_account_name = "terraform123abc"
    container_name       = "terraform-state"
    key                  = "prod.terraform.tfstate"
  }
}
```

Data Source Config Using Azure AD:

```
data "terraform_remote_state" "foo" {
  backend = "azurerm"
  config = {
    storage_account_name = "terraform123abc"
    container_name       = "terraform-state"
    key                  = "prod.terraform.tfstate"
    use_azuread_auth     = true
    subscription_id      = "00000000-0000-0000-0000-000000000000"
    tenant_id            = "00000000-0000-0000-0000-000000000000"
  }
}
```

Standard Backend: azurerm

A CLOUD GURU

Configuration Variables:

- `storage_account_name` (Required)
- `container_name` (Required)
- `key` (Required)
- `environment` (Optional)
- `endpoint` (Optional)
- `snapshot` (Optional)

Azure AD Config Variables:

- `use_azuread_auth` (Optional)

Service Principal Config Variables:

- `resource_group_name` (Required)
- `client_id` (Optional)
- `client_certification_password` (Optional)
- `client_certification_path` (Optional)
- `subscription_id` (Optional)
- `tenant_id` (Optional)
- `client_secret` (Optional)

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Backend Configuration

Standard Backend: consul

A CLOUD GURU

Example Block Configuration:

```
terraform {
  backend "consul" {
    address = "consul.example.com"
    scheme  = "https"
    path    = "full/path"
  }
}
```

Data Source Configuration:

```
data "terraform_remote_state" "foo" {
  backend = "consul"
  config = {
    path = "full/path"
  }
}
```

Configuration Variables:

- `path` (Required)
- `access_token` (Required)
- `address` (Optional)
- `scheme` (Optional)
- `datacenter` (Optional)
- `http_auth` (Optional)
- `gzip` (Optional)
- `lock` (Optional)
- `ca_file` (Optional)
- `cert_file` (Optional)
- `key_file` (Optional)

Standard Backend: etcd

A CLOUD GURU

Example Block Configuration:

```
terraform {
  backend "etcd" {
    path      = "path/to/terraform.tfstate"
    endpoints = "http://one:4001 http://two:4001"
  }
}
```

Data Source Configuration:

```
data "terraform_remote_state" "foo" {
  backend = "etcd"
  config = {
    path      = "path/to/terraform.tfstate"
    endpoints = "http://one:4001 http://two:4001"
  }
}
```

Configuration Variables:

- `path` (Required)
- `endpoints` (Required)
- `username` (Optional)
- `password` (Optional)

A CLOUD GURU

Working with State

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

State is a requirement for Terraform to function properly

- Terraform requires some sort of database to map Terraform configuration to the real world.
- Terraform uses its own state structure to map configurations to resources.

Ways Terraform Uses State

A CLOUD GURU

Metadata

Terraform must track metadata like resources dependencies. To ensure operation, Terraform retains a copy of the most recent set of dependencies within the state.

Performance

Terraform stores a cache of attribute values for all resources in the state. This comes in handy with large configurations, so you don't have to query all the resources in the configuration.

Syncing

When working in a team, it is recommended to use remote state. It is important that, if working in a team, all members work with the same state. Remote state ensures the state is synced.

State with Backend Configuration

A CLOUD GURU

Remote Backend

```
data "terraform_remote_state" "vpc" {
  backend = "remote"

  config = {
    organization = "hashicorp"
    workspaces = {
      name = "vpc-prod"
    }
  }
}

# Terraform >= 0.12
resource "aws_instance" "foo" {
  # ...
  subnet_id =
data.terraform_remote_state.vpc.outputs.subnet_id
}
...
```

Local Backend

```
data "terraform_remote_state" "vpc" {
  backend = "local"

  config = {
    path = "..."
  }
}

# Terraform >= 0.12
resource "aws_instance" "foo" {
  # ...
  subnet_id =
data.terraform_remote_state.vpc.outputs.subnet_id
}
...
```

State Locking

A CLOUD GURU

Things to Know:

- If supported by your backend of choice, Terraform locks your state for all operations.
- State locking happens automatically on all operations that could write state.
- Terraform outputs a status message if acquiring the lock is taking longer than expected.
- Not all backends support locking.

Force Unlock Command:

```
$ terraform force-unlock [options] LOCK_ID [DIR]
```

Force Unlock Tips:

- **Be careful with this command!** If you unlock the state when in use, it can cause multiple writer.
- To protect against this, the `force-unlock` command requires a unique lock ID.

10 Backends That Support Multiple Workspaces

1. AzureRM
2. Consul
3. COS
4. GCS
5. Kubernetes
6. Local
7. Manta
8. Postgres
9. Remote
10. S3

Remote state allows Terraform to write the state data to a remote data store, which can be shared between all team members.

1. Remote state allows you to share output values with other configurations.
2. Remote state allows teams to share infrastructure resources in a read-only way.
3. Remote state can be a convenient, built-in mechanism for sharing data between configurations, but you may prefer to use more general stores to pass settings to other configurations and consumers.

State CLI Commands

A CLOUD GURU

```
$ terraform state <subcommand> [option] [args]
```

Terraform State Syntax

```
$ terraform state mv packet.foo packet.bar
```

State Move

```
$ terraform state list
```

State List

```
$ terraform state rm 'packet.bar'
```

State Remove

```
$ terraform state show 'module.name.foo.worker'
```

State Show

```
$ terraform state pull
```

State Pull

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Working with State

A CLOUD GURU

Managing Workspaces

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Workspaces are separate instances of state data that can be used from the same working directory.

- Workspaces allow you to use the same working copy of your configuration as well as the same plugin and module caches, while still keeping separate states for each collection of resources you manage.

Workspaces

Workspaces are separate instances of **state data** that can be used from the same working directory. You can use **workspaces** to manage multiple non-overlapping groups of resources with the same configuration.

Things to Know:

- Every initialized working directory has at least one **workspace**.
- For a given working directory, only one **workspace** can be selected at a time.
- Most **Terraform** commands only interact with the currently selected **workspaces**.
- Use the `terraform workspace select` command to change the currently selected workspace.
- Use the `terraform workspace list`, `terraform workspace new`, and `terraform workspace delete` commands to manage the available **workspaces** in the current working directory.

Workspace Commands

A CLOUD GURU

workspace list

```
$ terraform workspace list
  default
* development
  jsmith-test
```

workspace select

```
$ terraform workspace select default
  Switched to workspace “default”.
```

workspace show

```
$ terraform workspace show
  development
```

workspace delete

```
$ terraform workspace delete example
  Deleted workspace “example”.
```

Workspace Commands

A CLOUD GURU

workspace new

```
$ terraform workspace new example
Created and switched to workspace "example"!

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.
```

workspace new: from state

```
$ terraform workspace new -state=old.terraform.tfstate example
Created and switched to workspace "example".

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.
```

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Managing Workspaces

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Terraform Fundamentals

SECTION RECAP

Terraform Fundamentals

- Terraform CLI
- Configuration Language
- Working with Resources
- Input Variables
- Declaring Output Variables
- Declaring Local Variables
- Modules

Jesse Hoch
Training Architect

A CLOUD GURU

SECTION RECAP CONTINUED

Terraform Fundamentals

Module Sources

Using Expressions and Functions

Backend Configuration

Working with States

Managing Workspaces

Labs:

- Import Terraform Configurations
- How to Work with Terraform Variables
- Use Terraform to Deploy Multiple Environments
- Build a Module for Terraform

Jesse Hoch
Training Architect

A CLOUD GURU

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Terraform Cloud

SECTION OVERVIEW

Terraform Cloud

Migrating Local Terraform

Integrations

Workspaces in Terraform Cloud

Runs in Terraform Cloud

Run States and Stages

Users, Teams, and Organizations

Section Recap

Labs:

- Migrate Terraform State to Terraform Cloud
- Connect Terraform Workspaces with Run Triggers in Terraform Cloud

Jesse Hoch
Training Architect

A CLOUD GURU

Migrating Local Terraform

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Migrating your Terraform state to Terraform Cloud lets you continue managing that infrastructure without de-provisioning anything.

- Terraform Cloud is an application that helps teams use Terraform together. It manages Terraform runs in a consistent and reliable environment.

Steps to Migrate Local Terraform

A CLOUD GURU

Step 1:

You will need Terraform 0.11.13 or later on the workstation you are doing the migration on. Older versions will either not have the remote backend, or will only partially support it.

Step 2: Gather Credentials, Data, and Code

Make sure you have the following:

- The location of the VCS repository for the Terraform config
- A local working copy of the Terraform config
- The existing state data
- A Terraform Cloud user account

Steps to Migrate Local Terraform

A CLOUD GURU

Step 3: Stop Terraform Runs

Let any current runs finish, then make sure there won't be more.
This could involve locking or deleting CI jobs, restricting access to the state backend, or just communicating clearly with your team.

Step 4: Prepare Your Terraform Working Directory

In your working directory:

- If you've already been doing Terraform runs in this directory, you should be good to go.
- If you had to retrieve a terraform.tfstate file from elsewhere, copy it into the root of the working directory and run `terraform init`.
- If you used a remote backend and had to check out a fresh working directory from version control, run `terraform init`.

Steps to Migrate Local Terraform

A CLOUD GURU

Step 5: Edit the Backend Config

```
terraform {
  backend "remote" {
    hostname = "app.terraform.io"
    organization = "my-org"

    workspaces {
      name = "my-workspace"
    }
  }
}
```

Values needed:

- Use the remote backend.
- In the `organization` attribute, specify the name of your Terraform Cloud organization.
- The `hostname` attribute is only necessary with Terraform Enterprise instances.
- In the `name` attribute within the workspaces block, specify the name of the new Terraform Cloud workspace to create with your state.

Steps to Migrate Local Terraform

A CLOUD GURU

Step 6: Run `terraform init` to Migrate the Workspace

```
$ terraform init
```

```
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 "remote" backend. No existing state
was found in the newly
configured "remote" backend. Do you want to copy this
state to the new "remote"
backend? Enter "yes" to copy and "no" to start with an
empty state.
```

Step 7: Configure the Terraform Cloud Workspace

Possible setting changes:

- Set the VCS repository
- Set values for variables
- Set team access permissions

Steps to Migrate Local Terraform

A CLOUD GURU

Step 8: Queue a Run in the New Workspace

Run `terraform plan` or navigate to the workspace in Terraform Cloud and queue a plan.

The plan should result in no to small changes. After that, Terraform Cloud can take over all Terraform runs for the infrastructure.

Troubleshooting

- If the plan would create an entirely new set of infrastructure resources, you probably have the wrong state file.
- If the plan recognizes the existing resources but would make unexpected changes, check whether the designated VCS branch for the workspace is the same branch you've been running Terraform on.

Steps to Migrate Multiple Local Workspaces

Steps 1 - 4 and Steps 7-8 are the same.

Step 5: Edit the Backend Config

```
terraform {
  backend "remote" {
    hostname = "app.terraform.io"
    organization = "my-org"

    workspaces {
      prefix = "my-app"
    }
  }
}
```

Step 6: Run `terraform init` to Migrate the Workspace

Backends like the default local backend allow special default workspaces that don't have a specific name. If you use a mix of these and named backends, you will be prompted to choose a new name.

```
The "remote" backend configuration only allows named
workspaces!
  Please provide a new workspace name (e.g. dev, test)
that will be used
  to migrate the existing default workspace.

  Enter a value:
```

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Migrating Local Terraform

A CLOUD GURU

Integrations

Jesse Hoch
TRAINING ARCHITECT

Terraform Cloud is more powerful when you integrate it with a version control system (VCS).

- When workspaces are linked to a VCS repository, Terraform Cloud can automatically initiate Terraform runs when changes are committed.
- Terraform Cloud makes code review easier by automatically predicting how pull requests will affect infrastructure.
- Publishing new versions of a private Terraform module is like pushing a tag to the module's repo.

9 Supported VCS Providers

1. GitHub.com
2. GitHub.com (OAuth)
3. GitHub Enterprise
4. GitLab.com
5. GitLab EE and CE
6. Bitbucket Cloud
7. Bitbucket Server
8. Azure DevOps Server
9. Azure DevOps Services

How VCS Access Is Used

Most workspaces in Terraform Cloud are associated with a VCS repository, which provides Terraform configurations for that workspace. To find out which repos are available, access their contents, and create webhooks, Terraform Cloud needs access to your VCS provider.

To use VCS configs, Terraform Cloud needs to do the following:

- Access a list of repositories
- Register webhooks with your VCS provider
- Download the contents of a repository at a specific commit

webhooks

- When someone adds new commits to a branch, any Terraform Cloud workspaces based on that branch will begin a Terraform run.
- When someone submits a pull request/merge request to a branch, any Terraform Cloud workspaces based on that branch will perform a speculative plan with the contents of the request and links to the results on the pull request's page.

3 Tips for Using SSH Keys

1. For most supported VCS providers, Terraform Cloud does not need an SSH key. The exceptions are Azure DevOps Server and Bitbucket Server, which require an SSH key for downloading repository contents.

2. For other VCS providers, most organizations will not need to add an SSH private key. However, if the organization repositories include Git submodules that can only be accessed via SSH, an SSH key can be added along with the OAuth credentials.

3. For VCS providers where adding an SSH private key is optional, SSH will only be used to clone Git submodules. All other Git operations will still use HTTPS.

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Integrations

A CLOUD GURU

Workspaces in Terraform Cloud

Jesse Hoch
TRAINING ARCHITECT

Workspaces are a collection of infrastructure.

- When run locally, Terraform manages each collection of infrastructure with a persistent working directory, which contains a configuration, state data, and variables.
- Terraform Cloud manages infrastructure collections with workspaces instead of directories.

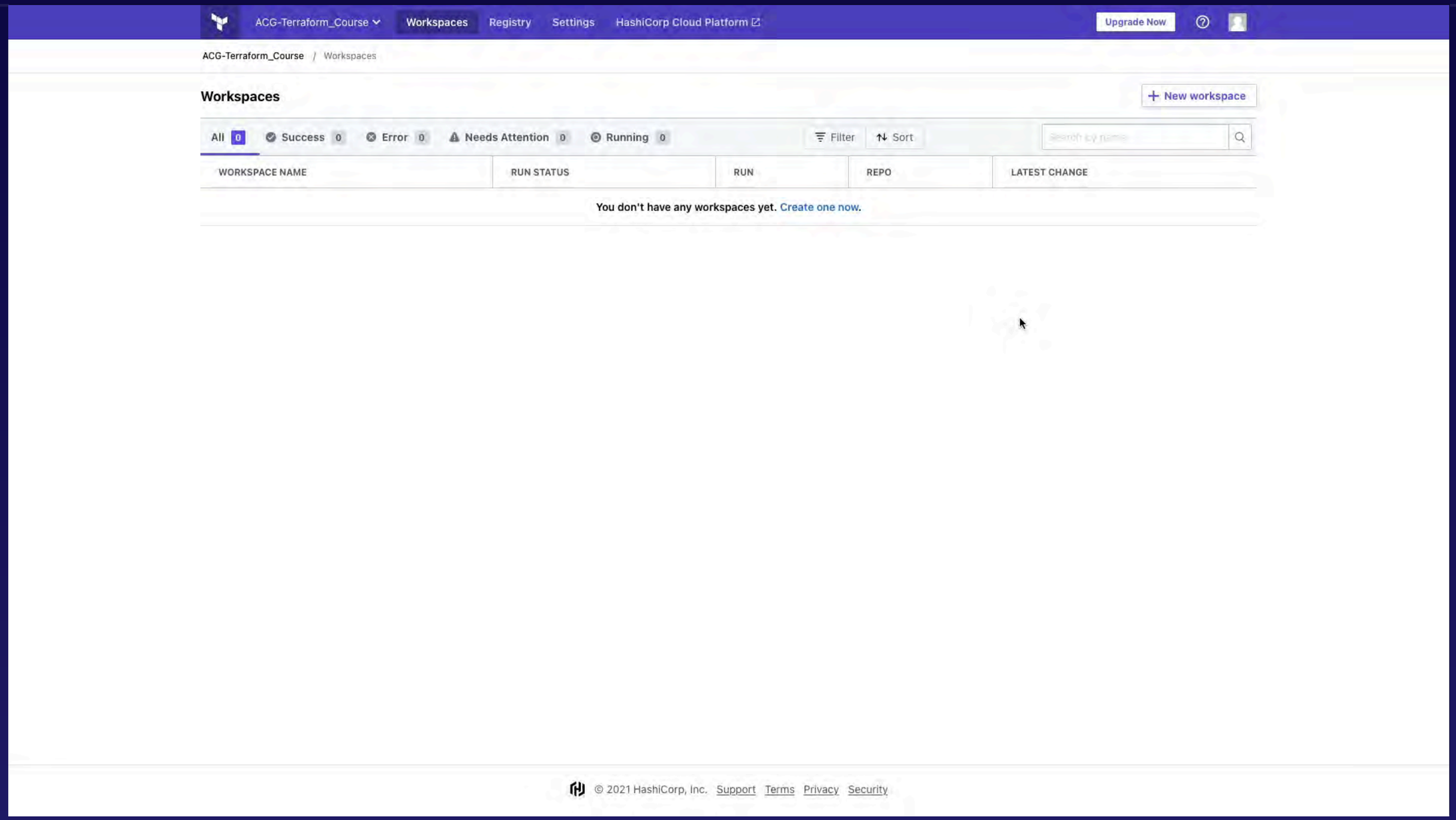
Workspace Contents

- **State versions:** Each workspace retains backups of its previous state files.
- **Run history:** When Terraform Cloud manages a workspace's Terraform runs, it retains a record of all run activity, including summaries, logs, a reference to the changes that caused the run, and user comments.

Component	Local Terraform	Terraform Cloud
Terraform Config	On disk	In VCS or uploaded via API/CLI
Variable Values	As .tfvars files, CLI arguments, or as shell environment	In workspaces
State	On disk or in remote backend	In workspaces
Credentials and Secrets	In shell environment or entered at prompts	In workspaces, stored as sensitive variables

Configuring a New Workspace

A CLOUD GURU



After Creating a Workspace

A CLOUD GURU

After creating a workspace:

- It presents a dialog with links to either queue a plan or edit variables.
- It registers a webhook with your VCS if you have connected a VCS repository to a workspace.

Most common things to do after creating a workspace:

- Edit variables
- Edit additional workspace settings
- Work with runs

Variables in Terraform Cloud

ACG-Terraform_Course | Workspaces | Registry | Settings | HashiCorp Cloud Platform | Upgrade Now

ACG-Terraform_Course / Workspaces / learn-terraform-github-actions / Variables

learn-terraform-github-actions

No workspace description available. Add workspace description.

Resources	Terraform version	Updated
0	1.0.0	a few seconds ago

Overview | Runs | States | Variables | Settings | Queue plan manually

Variables

These variables are used for all plans and applies in this workspace. Workspaces using Terraform 0.10.0 or later can also load default values from any `*.auto.tfvars` files in the configuration.

Sensitive variables are hidden from view in the UI and API, and can't be edited. (To change a sensitive variable, delete and replace it.) Sensitive variables can still appear in Terraform logs if your configuration is designed to output them.

When setting many variables at once, the Terraform Cloud Provider or the variables API can often save time.

Terraform Variables

These Terraform variables are set using a `terraform.tfvars` file. To use interpolation or set a non-string value for a variable, click its HCL checkbox.

Key	Value
There are no variables set.	

+ Add variable

Environment Variables

These variables are set in Terraform's shell environment using `export`.

Key	Value
There are no variables set.	

+ Add variable

3 Ways to Use Variables

1. **Terraform Variables** - **Terraform Cloud** passes variables to Terraform by writing a `terraform.tfvars` file and passing the `-var-file=terraform.tfvars` option to the Terraform command.

2. **Environment Variables** - **Terraform Cloud** performs Terraform runs on disposable Linux worker VMs. Before running **Terraform**, **Terraform Cloud** populates the shell with environment variables using the `export` command.

3. **Special Environment Variables** - Terraform Cloud uses these to control dangerous or rarely used run behaviors. If TFE_PARALLELISM is present, Terraform Cloud uses this to set `terraform plan` and `terraform apply`'s -parallelism=<N> flag.

Managing Run Triggers

A CLOUD GURU

ACG-Terraform_Course | Workspaces | Registry | Settings | HashiCorp Cloud Platform | Upgrade Now

ACG-Terraform_Course / Workspaces / learn-terraform-github-actions / Overview

learn-terraform-github-actions

No workspace description available. Add workspace description.

Resources: 0
Terraform version: 1.0.0
Updated: a few seconds ago

Overview | Runs | States | Variables | Settings

Queue plan manually

Latest Run View all runs

Triggered via UI ERRORED

Triggered by jessehoch78 2 minutes ago. From Terraform Cloud UI using master b990650

Policy checks	Estimated cost change	Plan duration	Resources to be changed
Upgrade	Upgrade	Less than a minute	+0 ~0 -0

See details

Outputs (0) Current as of the most recent state version.

NAME ↓	TYPE	VALUE
This workspace does not have any defined outputs.		

Learn about outputs

README.md

Automate Terraform with GitHub Actions

This repo is a companion repo to the Automate Terraform with GitHub Actions.

JesseHoch/learn-terraform-github-actions
Readme
Execution mode: Remote
Auto apply: Off

Metrics (last 2 runs)

Average plan duration	n/a
Average apply duration	n/a
Total failed runs	2
Policy check failures	Upgrade

Run triggers

No source workspaces have been selected. Adding run triggers will allow runs to queue automatically in this workspace.

Contributors (1)

0

Team management is a paid feature. Free organizations only include an owners team that can have up to five members. Learn more about Terraform Cloud pricing.

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Workspaces in Terraform Cloud

A CLOUD GURU

Runs in Terraform Cloud

Jesse Hoch
TRAINING ARCHITECT

Navigating Runs

Each workspace has two ways to view runs:

- A **Runs link**, which brings you to a full list of runs.
- A **Current Run link**, which takes you to the most recent active run.

The run page shows:

- The current status of the run.
- The code commit associated with the run.
- How, when, and by whom the run was initiated.
- A timeline of events related to the run.
- The output from both `terraform plan` and `apply` commands.

Destroy Mode

Destroy mode instructs **Terraform** to create a plan which destroys all objects, regardless of configuration changes.

- **CLI**: Use `terraform plan -destroy` or `terraform destroy`
- **API**: Use the `is-destroy` option
- **UI**: Use a workspace's **"Destruction and Deletion"** settings page

Refresh-Only Mode

Refresh-only mode instructs **Terraform** to create a plan which updates the **Terraform** state to match changes made to remote objects outside of **Terraform**.

- **CLI**: Use `terraform plan -refresh-only` or `terraform apply -refresh-only`
- **API**: Use the `refresh-only` option

Skipping Automatic State Refresh

The `-refresh=false` option is used in normal planning mode to skip the default behavior of the refreshing of the state before checking for configuration changes.

- **CLI**: Use `terraform plan -refresh=false` or `terraform apply -refresh=false`
- **API**: Use the `refresh` option

Replacing Selected Resources

The `replace` option instructs Terraform to replace the object with the given resource address.

- **CLI**: Use `terraform plan -replace=ADDRESS` or `terraform apply -replace=ADDRESS`
- **API**: Use the `replace-adds` option

Target Plan and Apply

Resource Targeting is intended for exceptional circumstances only and should not be used often.

- **CLI**: Use `terraform plan -target=ADDRESS` or `terraform apply -target=ADDRESS`
- **API**: Use the `target-addrs` option

There are limitations on **Terraform Cloud** features for remote plans created with targeting:

- **Sentinel policy checks** for targeted plans may cause a failure for any policy that requires a planned change to a resource instance elected by its address.
- **Cost Estimation** is disabled for any run created with `-target` set to prevent producing a misleading underestimate of cost due to resource instances being excluded from the plan.

3 Types of Run Workflows

1. **UI/VCS-driven Runs** - In this workflow, every workspace is associated with a specific branch of a VCS repo of Terraform configurations.

2. **API-driven Runs** - In this workflow, workspaces are not directly associated with a VCS repo, and runs are not driven by webhooks on your VCS provider.

3. **CLI-driven Runs** - **Terraform Cloud** runs **Terraform CLI** to provision infrastructure.

Terraform Cloud's Run Environment

A CLOUD GURU

Terraform Worker VMs	Terraform Cloud performs runs in single-use Linux VMs.
Network Access to VCS and Infrastructure Providers	Terraform Cloud needs access to all resources being managed in order to perform runs.
Concurrency and Run Queuing	Terraform Cloud uses multiple concurrent worker VMs. They take jobs from a queue of runs that are ready for processing.
State Access and Authentication	Terraform Cloud stores state for its workspaces.

Environment Variables:

- TFC_RUN_ID
- TFC_WORKSPACE_NAME
- TFC_WORKSPACE_SLUG
- TFC_CONFIGURATION_VERSION_GIT_BRANCH
- TFC_CONFIGURATION_VERSION_GIT_COMMIT_SHA
- TFC_CONFIGURATION_VERSION_GIT_TAG

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Runs in Terraform Cloud

A CLOUD GURU

Run States and Stages

Jesse Hoch
TRAINING ARCHITECT

6 Stages of Run State

1. Pending Stage
2. Plan Stage
3. Cost Estimation Stage
4. Policy Check Stage
5. Apply Stage
6. Completion

The Pending Stage

A CLOUD GURU

States in this stage:

- **Pending: Terraform Cloud** hasn't started action on any run yet. It processes each workspace's runs in the order they were queued.

Leaving this stage:

- Proceeds automatically to the plan stage when it becomes the first run in the queue.
- Can skip to completion if a user discards it before it starts.

The Plan Stage

A CLOUD GURU

States in this stage:

- **Planning: Terraform Cloud** is currently running `terraform plan`.
- **Needs Confirmation: `terraform plan`** has finished. Runs sometimes pause in this state, depending on the workspace and organization settings.

Leaving this stage:

- If the `terraform plan` command failed, the run skips to completion.
- If a user canceled the plan by pressing the "**Cancel Run**" button, the run skips to completion.
- If the plan succeeded and doesn't require any changes, the run skips to completion.
- If the plan succeeded and requires changes, it will proceed to the proper stage.

The Cost Estimation Stage

A CLOUD GURU

States in this stage:

- **Cost Estimating:** Terraform Cloud is currently estimating the resources in the plan.
- **Cost Estimated:** The cost estimate completed.

Leaving this stage:

- If the cost estimation succeeded or errors, the run moves to the next stage.
- If there are no policy checks or applies, the run skips to completion.

The Policy Check Stage

A CLOUD GURU

States in this stage:

- **Policy Check: Terraform Cloud** is currently checking the plan against the organization's policies.
- **Policy Override:** The policy check finished. If a policy fails, an apply can't proceed without approval from a user who has permissions to manage policy overrides.
- **Policy Checked:** The policy check succeeded, and Sentinel will allow an apply to proceed.

Leaving this stage:

- If any hard-mandatory policies failed, the run skips to completion.
- If any soft-mandatory policies failed, the run pauses in the Policy Override state.
- If the run reaches the Policy Checked state:
  - If the plan can be auto-applied, the run proceeds to the apply stage.
  - If the plan can't be auto-applied, the run pauses in the Policy Checked state until a user with permissions to apply runs takes action.

The Apply Stage

A CLOUD GURU

States in this stage:

- **Applying:** Terraform Cloud is currently running `terraform apply`.

Leaving this stage:

- If the apply succeeded, the run ends in the Applied state.
- If the apply failed, the run ends in the Apply Errored state.
- If the user canceled the apply by pressing the "Cancel Run" button, the run ends in the Canceled state.

Completion

States in this stage:

- **Applied:** The run was successfully applied.
- **Planned:** `terraform plan`'s output already matches the current infrastructure state, so `terraform apply` doesn't need to do anything.
- **Apply Errored:** The `terraform apply` command failed, possibly due to a missing or misconfigured provider or an illegal operation on a provider.
- **Planned Errored:** The `terraform plan` command failed, or a hard-mandatory Sentinel policy failed. The run cannot be applied.
- **Discarded:** A user chose not to continue this run.
- **Canceled:** A user interrupted the `terraform plan` or `terraform apply` command with the "Cancel Run" button.

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Run State and Stages

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Terraform Cloud

SECTION RECAP

Terraform Cloud

Migrating Local Terraform

Integrations

Workspaces in Terraform Cloud

Runs in Terraform Cloud

Run States and Stages

Users, Teams, and Organizations

Labs:

- Migrate Terraform State to Terraform Cloud
- Connect Terraform Workspaces with Run Triggers in Terraform Cloud

Jesse Hoch
Training Architect

A CLOUD GURU

Users, Teams, and Organizations

Jesse Hoch
TRAINING ARCHITECT

Organizational and Access Control Model

Terraform Cloud's organizational and access control model is based on three units:

Users

Individual members of an organization. They belong to teams, which are granted permissions on an organization's workspaces.

Teams

Groups of users that reflect your company's organizational structure.

Organizations

Shared spaces for teams to collaborate on workspaces. An organization can have many teams.

Users

Creating an Account:

- For the SaaS version of Terraform Cloud, go to https://app.terraform.io/signup/account.
- For Terraform Enterprise, go to https://<TFE_Hostname>/signup/account.

After Creating an Account:

- If you are the first Terraform Cloud user in your organization, use this page to create your Terraform Cloud organization.
- If you intend to join an existing organization, do not create a new one. Instead, send your email address to one of your organization's owners and ask them to invite you. Once you receive and accept the emailed invitation, you can reload the page to begin using Terraform Cloud.

Teams

The Owners Team:

- Creating and deleting items
- Inviting users to the organization
- Managing team membership and organization-level permissions granted to teams
- Viewing the full list of teams, both visible and secret
- Managing organization settings

Teams

Creating and Deleting Teams:

- To create a new team, provide a unique name and click **Create team**.
- To delete a team, go to the target team's settings page and click **Delete [TEAM NAME]**.

Managing Team Members:

- To add a user to a team, choose an existing organization member from the **Select user** drop-down menu.
- To add a teammate who doesn't yet belong to the organization, invite them to join the organization and include a list of teams they should belong to in the invitation. Once the invitation is accepted, they will be automatically added to those teams.
- To remove a user from the team, just click the **trash can** icon by their name in the members list.

Teams

Team Visibility:

- When a team is set to **Visible**, every user in the org can see the team and its members.
- When a team is set to **Secret**, only team members and org owners can read a team and its membership.

Managing Workspace Access:

- Use any workspace's **Access** tab to manage team permissions on that workspace.
- Set the appropriate team permission levels on the workspace.

Managing Organization Access:

- You can manage a teams org-level permissions on the team's settings page under **Organization Access**.
- Set the appropriate team permission levels for the organization.

Organizations

Joining and Leaving Organizations:

- To join an organization, you must be invited by one of its owners.
- You can leave an organization from your user account settings.

Creating Organizations:

- Users can create new organizations by clicking Create new organization in the organization switcher menu.
- To create a new organization, you must provide a unique name and primary contact email address.

Organizations

Organization Settings:

- Users with permissions to manage VCS settings or policies can also manage the settings they have access to.
- Organization owners can view and manage the entire list of org settings.
- Other users can use this section to view the organization's contact email, view the membership of any teams they belong to, and view the organization's auth policy.

Renaming an Organization:

- Alert all members of the organization ahead of time.
- Lock all workspaces and bring all current and pending runs to an end.
- Rename the organization.
- Update anything that uses Terraform Cloud's API with the organization to use its new name.
- Unlock workspaces and resume normal operations.

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Users, Teams, and Organizations

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Using Terraform in a CI/CD Environment

DEMO OVERVIEW

Using Terraform in a CI/CD Environment

Using GitHub Actions to Automate Terraform

Terraform Cloud Setup

GitHub Repository Setup

Actions Workflow

Creating and Merging Pull Requests

Verifying a Provisioned Instance

Section Recap

Labs:

- Import Terraform Configurations

Jesse Hoch
Training Architect

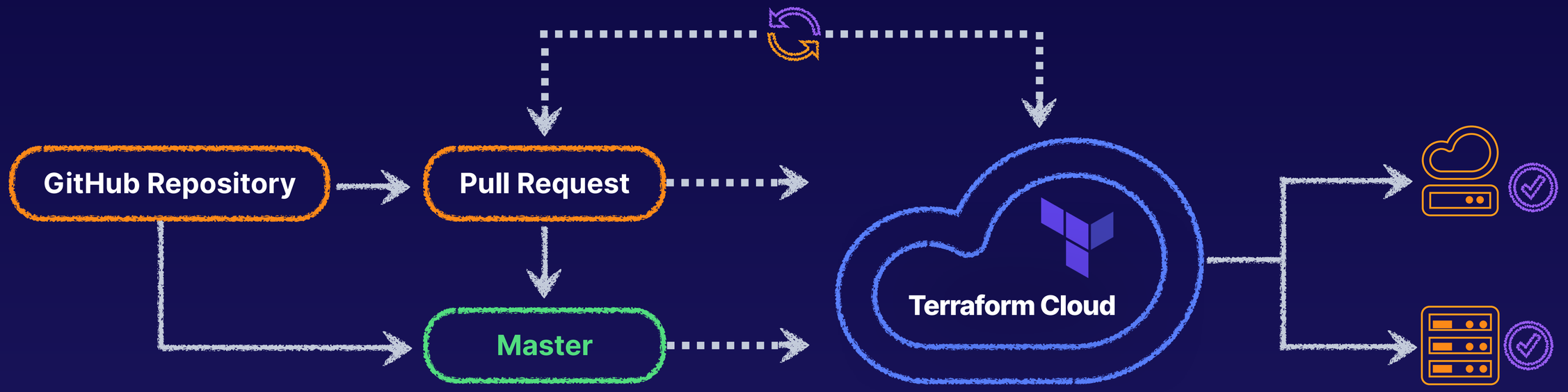
A CLOUD GURU

A CLOUD GURU

Demo: Using GitHub Actions to Automate Terraform

Jesse Hoch
TRAINING ARCHITECT

Demo Diagram



A CLOUD GURU

Demo Workflow:

- Check whether the configuration is formatted properly.
- Generate a plan for every pull request.
- Apply the configuration when you update the main branch.
- Test the workflow by creating and merging a pull request.

A CLOUD GURU

Prerequisites for this Demo:

- GitHub account
- Terraform Cloud account
- AWS account and AWS access credentials

A CLOUD GURU

Section Objectives:

- Terraform Cloud Setup
- GitHub Repository Setup
- Actions Workflow
- Creating and Merging Pull Requests
- Verifying a Provisioned Instance

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Using GitHub Actions to Automate Terraform

A CLOUD GURU

Demo: Terraform Cloud Setup

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Prerequisites for this Demo:

- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY

A CLOUD GURU

Demo Objectives:

- Create a Terraform Cloud organization and workspace
- Add environment variables
- Create an API token
- Save the token for later

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Terraform Cloud Setup

A CLOUD GURU

Demo: GitHub Repository Setup

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Prerequisites for this Demo:

- GitHub login: https://github.com/
- GitHub repository to clone: https://github.com/JesseHoch/learn-terraform-github-actions

A CLOUD GURU

Demo Objectives:

- Log in to GitHub and fork the provided repo
- Create a new secret named TF_API_TOKEN
- Use the Terraform API token from the previous demo
- Enable the pre-configured workflow
- Clone the Terraform GitHub Actions repository

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: GitHub Repository Setup

A CLOUD GURU

Demo: Actions Workflow

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Prerequisites for this Demo:

- Terraform GitHub Actions cloned repository

A CLOUD GURU

Demo Objectives:

- View the `.github/workflows/terraform.yml` file
- Review the `main.tf` configuration file

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Actions Workflow

A CLOUD GURU

Demo: Creating and Merging Pull Requests

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Prerequisites for this Demo:

- Terraform Cloud Organization
- Terraform Cloud Workspace

A CLOUD GURU

Demo Objectives:

- Create a new branch in your repo called `update-tfc-backend`.
- Update the `main.tf` file with your Terraform Cloud organization and workspace.
- Add, commit, and push your changes.
- Review and merge the pull request.

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Creating and Merging Pull Requests

A CLOUD GURU

Demo: Verifying a Provisioned Instance

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Demo Objectives:

- Check both the GitHub Actions and Terraform Cloud to verify the EC2 instance has been provisioned.
- Verify the EC2 instance is publicly available.
- Destroy the resources.

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Verifying a Provisioned Instance

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Using Using GitHub Actions to Automate Terraform

SECTION RECAP

Using GitHub Actions to Automate Terraform

- Using GitHub Actions to Automate Terraform
- Terraform Cloud Setup
- GitHub Repository Setup
- Actions Workflow
- Creating and Merging Pull Requests
- Verifying a Provisioned Instance

Labs:

- Build a CI/CD Pipeline using Terraform and GitHub

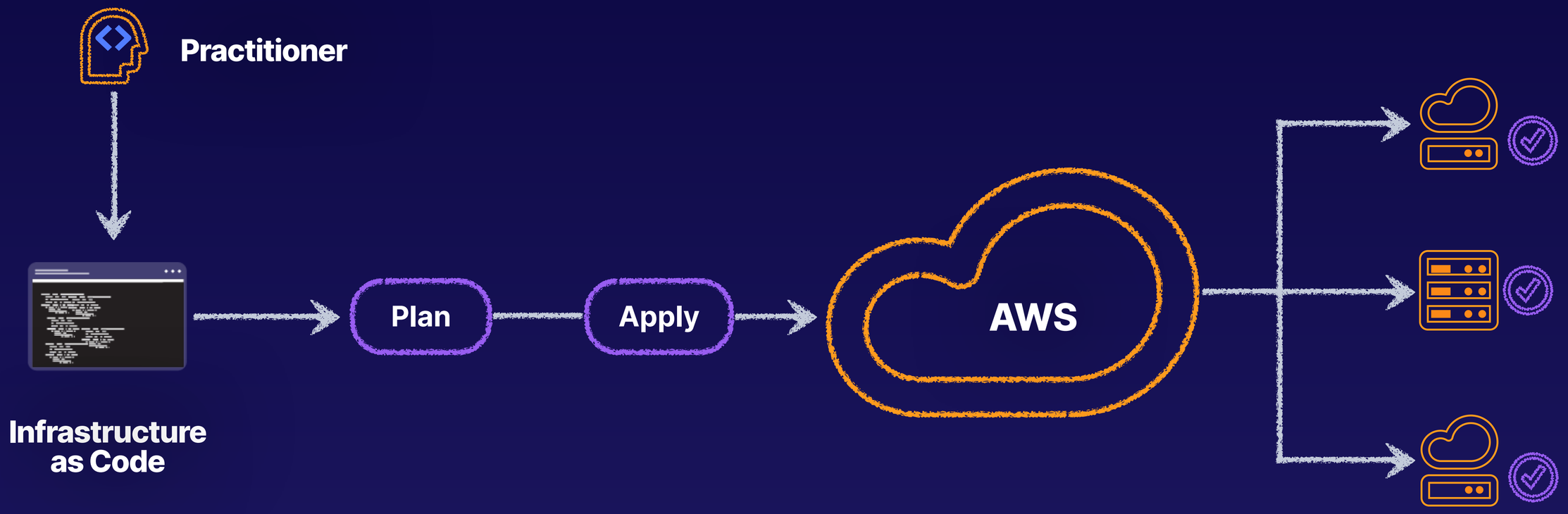
Jesse Hoch
Training Architect

A CLOUD GURU

Demo: Using Terraform to Create and Manage Infrastructure on AWS

Jesse Hoch
TRAINING ARCHITECT

Demo Diagram



A CLOUD GURU

Prerequisites for this Demo:

- Terraform CLI Installed
- AWS CLI Installed
- ACG's AWS sandbox set up and running
- Have your AWS credentials

A CLOUD GURU

Section Demo Objectives:

- Authentication
- Building Your Infrastructure
- Making Changes and Destroying Your Infrastructure
- Defining Input Variables
- Using Output Variables to Query Data
- Storing Remote State

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Using Terraform to Create and Manage Infrastructure on AWS

A CLOUD GURU

Demo: Authentication

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Prerequisites for this Demo:

- AWS CLI
- ACG's AWS sandbox
- AWS credentials

A CLOUD GURU

Demo Objectives:

- Install the AWS CLI
- Set up your AWS sandbox in ACG's Playground
- Configure the AWS CLI to use your AWS credentials

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Authentication

A CLOUD GURU

Demo: Building Your Infrastructure

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Prerequisites for this Demo:

- Install Terraform CLI
- The `main_aws.tf` file from our repo
- Make sure the AWS sandbox environment is set up and running

A CLOUD GURU

Demo Objectives:

- Create the `demo-terraform-aws-instance` directory
- Download the `main_aws.tf` file from the GitHub repo
- Initialize the working directory
- Format and validate the configuration
- Create the infrastructure
- Inspect the state

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Building Your Infrastructure

A CLOUD GURU

Demo: Making Changes and Destroying Your Infrastructure

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Prerequisites for this Demo:

- Terraform CLI installed
- AWS CLI installed
- Have the `main_aws.tf` file
- Make sure the AWS sandbox environment is set up and running

A CLOUD GURU

Demo Objectives:

- Change the AMI in the `main.tf` file
- Validate the change being made
- Apply the change to your infrastructure
- Do a dry run of the infrastructure deletion
- Destroy your infrastructure

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Making Changes and Destroying Your Infrastructure

A CLOUD GURU

Demo: Defining Input Variables

Jesse Hoch
DEVOPS TRAINING ARCHITECT

A CLOUD GURU

Prerequisites for this demo:

- Terraform CLI installed
- AWS CLI installed
- The main_aws.tf file
- AWS Sandbox environment setup and running

A CLOUD GURU

Section Demo Objectives:

- Create a variables.tf file.
- Within that file define the instance_name variable.
- Update the aws_instance resource block in the main.tf file with the aws_instance variable.
- Create your infrastructure and confirm its creation.

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Defining Input Variables

A CLOUD GURU

Demo: Defining Input Variables

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Prerequisites for this Demo:

- Install Terraform CLI
- Install AWS CLI
- Have the `main_aws.tf` file
- Make sure the AWS sandbox environment is set up and running

A CLOUD GURU

Demo Objectives:

- Create a `variables.tf` file
- Within that file, define the `instance_name` variable
- Update the `aws_instance` resource block in the `main.tf` file with the `aws_instance` variable
- Create your infrastructure and confirm its creation

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Defining Input Variables

A CLOUD GURU

Demo: Using Output Variables to Query Data

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Prerequisites for this Demo:

- Install Terraform CLI
- Install AWS CLI
- Have the `main_aws.tf` file
- Make sure the AWS sandbox environment is set up and running
- Have the `outputs_aws.tf` file

A CLOUD GURU

Demo Objectives:

- Download `outputs_aws.tf` and rename it `outputs.tf`
- Review the `outputs.tf` file
- Apply the configuration and inspect the output values
- Destroy the infrastructure

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Using Output Variables to Query Data

A CLOUD GURU

Demo: Storing Remote State

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Prerequisites for this Demo:

- Terraform Cloud
- Install Terraform CLI
- Install AWS CLI
- Have the `main_aws.tf` file
- Make sure the AWS sandbox environment is set up and running

A CLOUD GURU

Demo Objectives:

- Configure the remote backend
- Log in to Terraform Cloud
- Initialize Terraform and delete the state file
- Set the AWS credential workspace variables in Terraform Cloud
- Apply the configuration

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Storing Remote State

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Terraform and AWS

SECTION RECAP

Terraform and AWS

Using Terraform to Create and Manage Infrastructure on AWS

Authentication

Building Your Infrastructure

Making Changes and Destroying Your Infrastructure

Defining Input Variables

Using Output Variables to Query Data

Storing Remote State

Jesse Hoch
Training Architect

A CLOUD GURU

SECTION RECAP CONTINUED

Terraform and AWS

Labs:

- Deploy EC2 Instances with Terraform
- Make Changes to AWS Infrastructure Using Terraform
- Use Output Variables to Query Data in AWS Using Terraform

Jesse Hoch
Training Architect

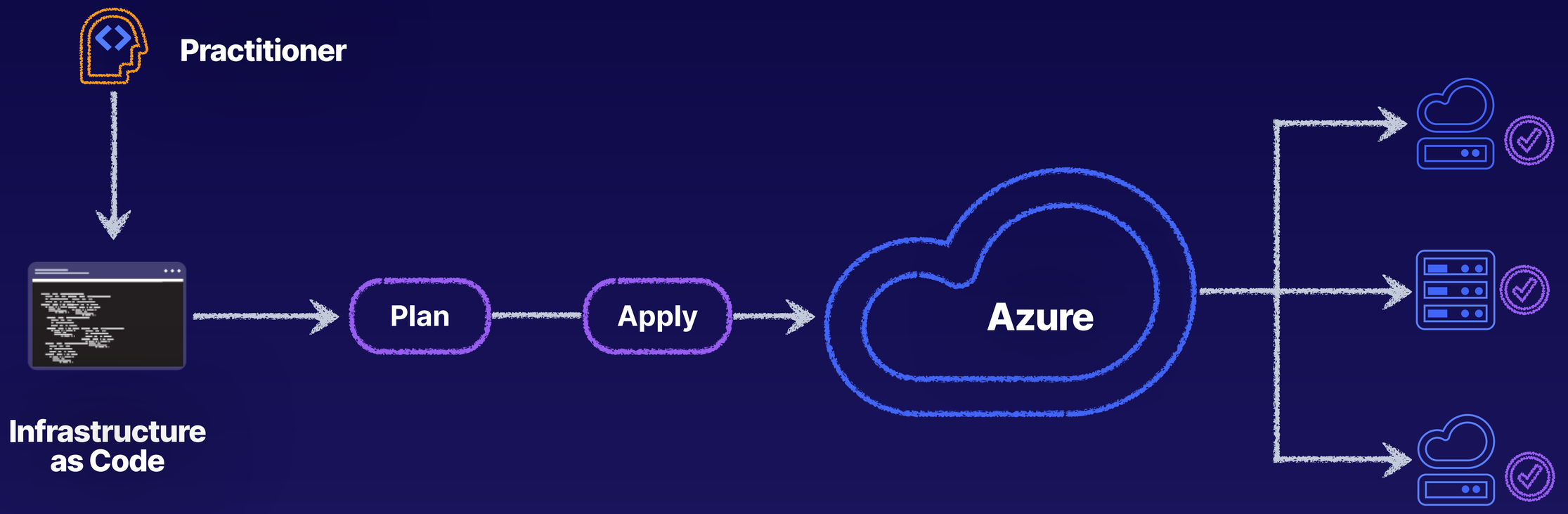
A CLOUD GURU

A CLOUD GURU

Demo: Using Terraform to Create and Manage Infrastructure on Azure

Jesse Hoch
TRAINING ARCHITECT

Demo Diagram



A CLOUD GURU

Prerequisites for this Demo:

- Install Terraform CLI
- Install Azure CLI
- Set up ACG's Azure sandbox
- Have Azure credentials

A CLOUD GURU

Section Demo Objectives:

- Authentication
- Building Your Infrastructure
- Making Changes and Destroying Your Infrastructure
- Defining Input Variables
- Using Output Variables to Query Data
- Storing Remote State

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Using Terraform to Create and Manage Infrastructure on Azure

A CLOUD GURU

Demo: Authentication

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Prerequisites for this Demo:

- Azure CLI
- ACG's Azure sandbox
- Azure credentials

A CLOUD GURU

Demo Objectives:

- Install and set up Azure CLI
- Set up your Azure sandbox in ACG's Playground
- Log in to the Azure sandbox using Azure CLI

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Authentication

A CLOUD GURU

Demo: Building Your Infrastructure

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Prerequisites for this Demo:

- Install Terraform CLI
- Install Azure CLI
- Have the `main_azure.tf` file
- Make sure the Azure sandbox environment is set up and running

A CLOUD GURU

Demo Objectives:

- Create the `demo-terraform-azure-instance` directory
- Download the `main_azure.tf` file from the GitHub repo
- Initialize the working directory
- Format and validate the configuration
- Create the infrastructure
- Inspect the state

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Building Your Infrastructure

A CLOUD GURU

Demo: Making Changes and Destroying Your Infrastructure

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Prerequisites for this Demo:

- Install Terraform CLI
- Install Azure CLI
- Have the `main_azure.tf` file
- Make sure the Azure sandbox environment is set up and running
- Have the `azure_resource_block.txt` file

A CLOUD GURU

Demo Objectives:

- Download the `azure_resource_block.txt` file from the GitHub repo
- Add the resource blocks from the file to the `main.tf` file
- Validate the change being made
- Apply the change to your infrastructure
- Do a dry run of the infrastructure deletion
- Destroy your infrastructure

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Making Changes and Destroying Your Infrastructure

A CLOUD GURU

Demo: Defining Input Variables

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Prerequisites for this Demo:

- Install Terraform CLI
- Install Azure CLI
- Have the `main_azure.tf` file
- Make sure the Azure sandbox environment is set up and running
- Have the `variables_azure.tf` file

A CLOUD GURU

Demo Objectives:

- Download the `variables_azure.tf` file from the GitHub repo into your working directory
- When downloading the file we will rename it variables.tf
- Update the `azurerm_subnet` resource block in the `main.tf` file with the variable
- Apply and confirm the configuration

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Defining Input Variables

A CLOUD GURU

Demo: Using Output Variables to Query Data

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Prerequisites for this Demo:

- Install Terraform CLI
- Install Azure CLI
- Have the `main_azure.tf` file
- Make sure the Azure sandbox environment is set up and running
- Have the `outputs_azure.tf` file

A CLOUD GURU

Demo Objectives:

- Download `outputs_azure.tf` and rename it `outputs.tf`
- Review the `outputs.tf` file
- Apply the configuration
- Inspect the output

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Using Output Variables to Query Data

A CLOUD GURU

Demo: Storing Remote State

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Prerequisites for this Demo:

- Terraform Cloud
- Install Terraform CLI
- Install Azure CLI
- Have the `main_azure.tf` file
- Make sure the Azure sandbox environment is set up and running

A CLOUD GURU

Demo Objectives:

- Configure the remote backend
- Log in to Terraform Cloud
- Initialize Terraform and delete the state file
- Configure a Service Principal
- Set the Azure credential workspace variables in Terraform Cloud
- Apply the configuration

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Storing Remote State

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Terraform and Azure

SECTION RECAP

Terraform and Azure

Using Terraform to Create and Manage Infrastructure on Azure

Authentication

Building Your Infrastructure

Making Changes and Destroying Your Infrastructure

Defining Input Variables

Using Output Variables to Query Data

Jesse Hoch
Training Architect

A CLOUD GURU

SECTION RECAP CONTINUED

Terraform and Azure

Labs:

- Deploy Azure VM Instance with Terraform
- Make Changes to Azure Infrastructure Using Terraform
- Use Output Variables to Query Data in Azure Using Terraform

Jesse Hoch
Training Architect

A CLOUD GURU

Demo: Using Terraform to Create and Manage Infrastructure on Kubernetes

Jesse Hoch
TRAINING ARCHITECT

A CLOUD GURU

Why Deploy with Terraform:

- Unified Workflow
- Full Lifecycle Management
- Graph of Relationships

A CLOUD GURU

Prerequisites for this Demo:

- Install Terraform CLI
- Install AWS, Azure, and kind CLIs
- Set up ACG's Azure and AWS sandbox
- Have credentials

A CLOUD GURU

Section Demo Objectives:

- Provision a Local Kubernetes cluster
- Provision an EKS cluster
- Provision an AKS cluster
- Manage Kubernetes with Terraform

Jesse Hoch

TRAINING ARCHITECT

A CLOUD GURU

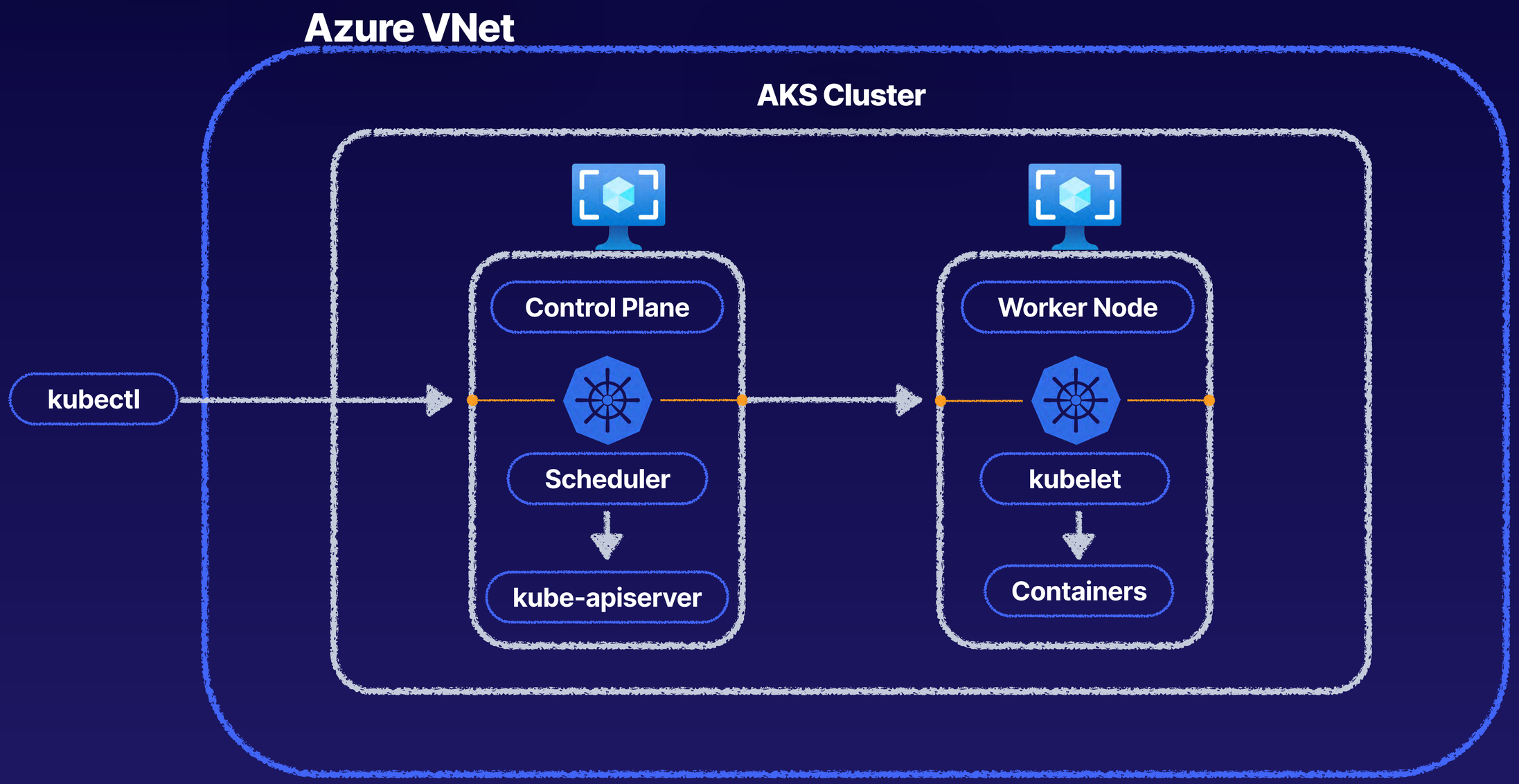
Demo: Using Terraform to Create and Manage Infrastructure on Kubernetes

A CLOUD GURU

Demo: Managing AKS with Terraform

Jesse Hoch
TRAINING ARCHITECT

Demo Diagram



A CLOUD GURU

Prerequisites for this Demo:

- Install Terraform CLI
- Install Azure CLI
- Install kubectl
- Have the `kubernetes_aks.tf` file
- Have the `service_aks_nginx.tf` file

A CLOUD GURU

Demo Objectives:

- Download the `kubernetes_aks.tf` file from the repo to your working directory and rename it to `kubernetes.tf`
- Download the `service_aks_nginx.tf` file from the repo and copy its contents to the `kubernetes.tf` file
- Download the `output_aks.tf` file from the repo and copy its contents to the `kubernetes.tf` file
- Apply the configuration to schedule the NGINX deployment
- Verify the NGINX deployment is running

A CLOUD GURU

Demo Objectives:

- Scale the deployment from 2 to 4 by changing the `replicas` field in the `kubernetes.tf` file
- Apply the change to scale the deployment
- Verify the scaling was successful
- Clean up the workspace

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Managing AKS with Terraform

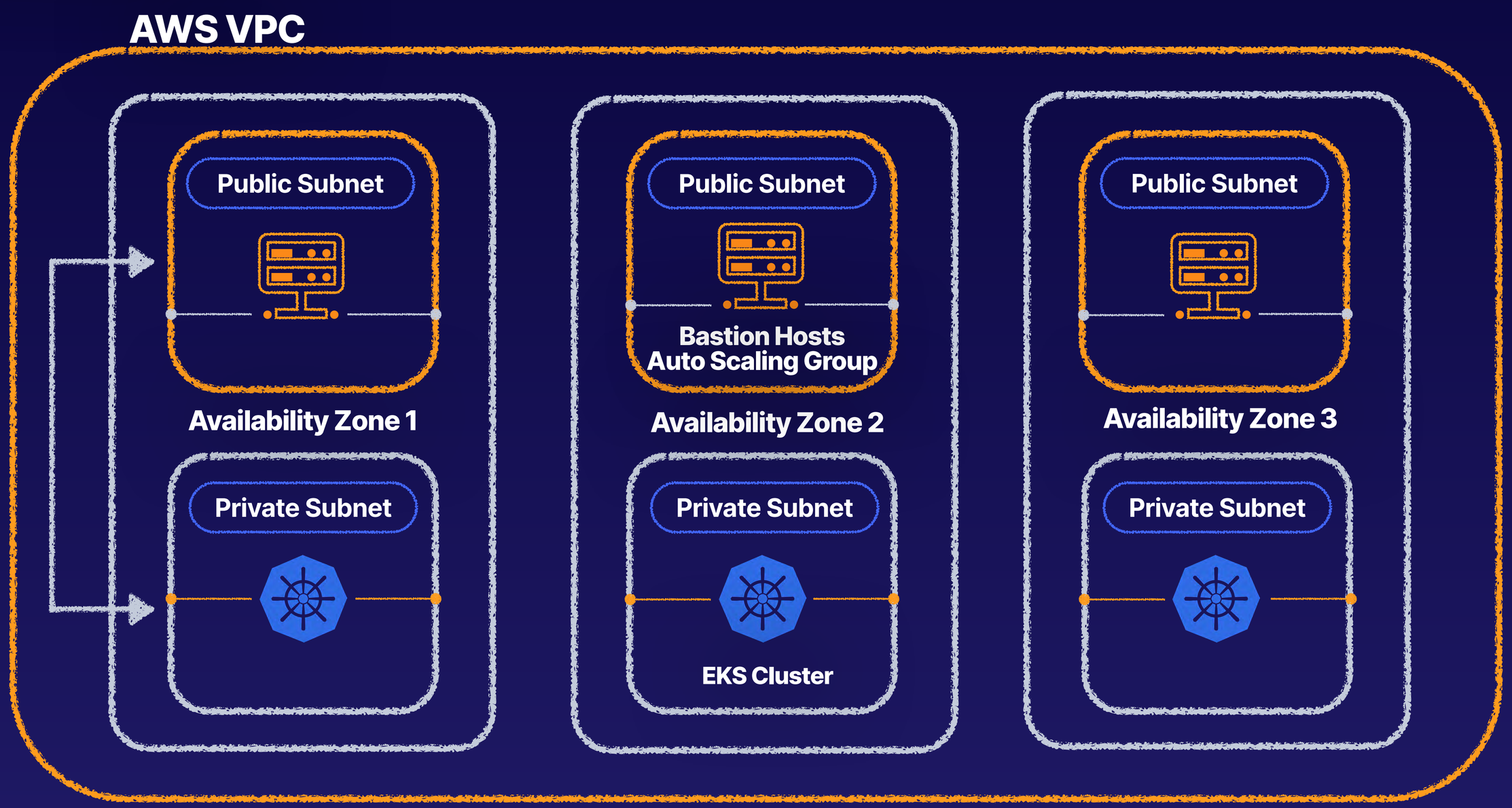
A CLOUD GURU

Demo: Managing EKS with Terraform

Jesse Hoch
TRAINING ARCHITECT

Demo Diagram

A CLOUD GURU



A CLOUD GURU

Prerequisites for this Demo:

- Install Terraform CLI
- Install AWS CLI
- Install kubectl
- Have the `kubernetes_eks.tf` file
- Have the `resource_nginx.tf` file
- Have the `service_eks_nginx.tf` file
- Have the `outputs_eks.tf` file

A CLOUD GURU

Demo Objectives:

- Create the demo-terraform-manage-eks working directory
- Download the `kubernets_eks.tf` file from the repo into your working directory and rename it kubernetes.tf
- View the `kubernetes.tf` file
- Apply the configuration
- Verify that EKS cluster is running

A CLOUD GURU

Demo Objectives:

- Download the `resource_nginx.tf` file from the repo into your working directory
- View the `resource_nginx.tf` file
- Apply the configuration to schedule the NGINX deployment
- Verify the NGINX deployment is running

A CLOUD GURU

Demo Objectives:

- Download the `service_eks_nginx.tf` file from the repo into your working directory
- View the `service_eks_nginx.tf` file
- Download the `outputs_eks.tf` file from the repo into your working directory
- View the `outputs_eks.tf` file
- Apply the configuration to schedule the LoadBalancer deployment
- Verify the LoadBalancer deployment is running

A CLOUD GURU

Demo Objectives:

- Scale the deployment from 2 to 4 by changing the `replicas` field in the `resource_nginx.tf` file
- Apply the change to scale the deployment
- Verify the scaling was successful
- Clean up the workspace

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Managing EKS with Terraform

A CLOUD GURU

Demo: Managing Kubernetes with Terraform

Jesse Hoch
TRAINING ARCHITECT

Demo Diagram

A CLOUD GURU

Prerequisites for this Demo:

- Install Terraform CLI
- Install kind CLI
- Have the `kind-config.yaml` file
- Have the `kubernetes.tf` file
- Have the `resource_nginx.tf` file

A CLOUD GURU

Demo Objectives:

- Download the `resource_nginx.tf` file from the repo
- Add the NGINX resources to the `kubernetes.tf` file
- Apply the configuration to schedule the NGINX deployment
- Verify the NGINX deployment is running

A CLOUD GURU

Demo Objectives:

- Download the `service_nginx.tf` file from the repo
- Add the NodePort service to the `kubernetes.tf` file
- Apply the configuration to schedule the NodePort service
- Verify the NGINX deployment is running

A CLOUD GURU

Demo Objectives:

- Scale the deployment from 2 to 4 by changing the `replicas` field in the `kubernetes.tf` file
- Apply the change to scale the deployment
- Verify the scaling was successful
- Clean up the workspace

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Managing Kubernetes with Terraform

A CLOUD GURU

Demo: Provisioning a Local Kubernetes Cluster

Jesse Hoch
TRAINING ARCHITECT

Demo Diagram

A CLOUD GURU

Prerequisites for this Demo:

- Install Terraform CLI
- Install kind CLI
- Have the `kind-config.yaml` file
- Have the `kubernetes.tf` file

A CLOUD GURU

Demo Objectives:

- Install kind
- Create the `demo-terraform-kubernetes` working directory
- Download the `kind-config.yaml` file
- Setup our local Kubernetes cluster
- Download the `kubernetes.tf` file
- Configure the Terraform Kubernetes provider

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Provisioning a Local Kubernetes Cluster

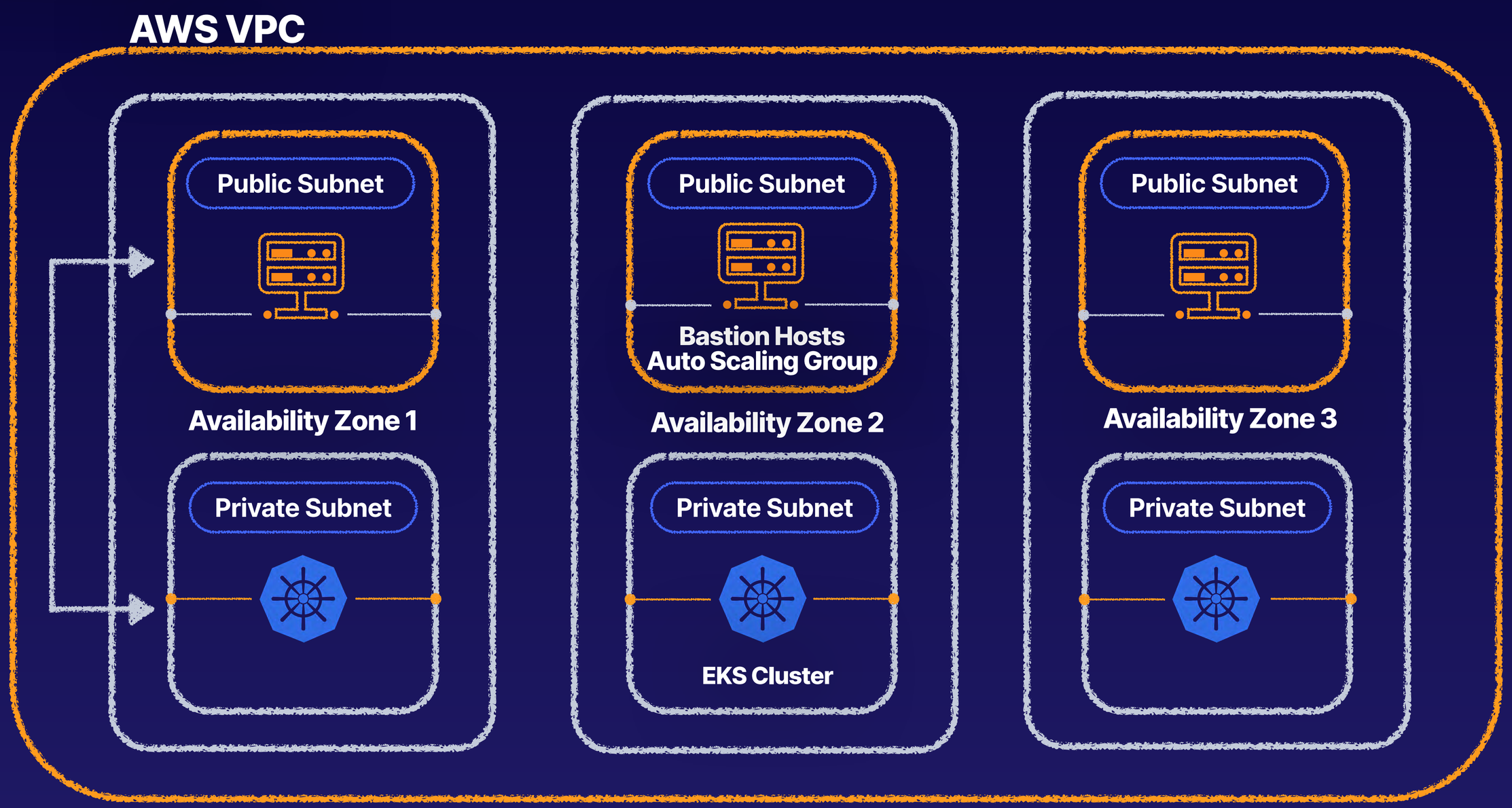
A CLOUD GURU

Demo: Provisioning an EKS Cluster

Jesse Hoch
TRAINING ARCHITECT

Demo Diagram

A CLOUD GURU



A CLOUD GURU

Prerequisites for this Demo:

- Install Terraform CLI
- Install AWS CLI
- Have ACG's AWS sandbox and credentials
- Have the `demo-terraform-eks-cluster.zip` file

A CLOUD GURU

Demo Objectives:

- Download the `demo-terraform-eks-cluster.zip` file from the repo
- Unzip the `demo-terraform-eks-cluster.zip` file and then cd into the directory
- Initialize the working directory
- Apply the configuration

A CLOUD GURU

Demo Objectives:

- Configure `kubectl`
- Download the `Kubernetes Metrics Server` tar file
- Unzip the file and apply it using `kubectl`
- Deploy the Kubernetes dashboard
- Authenticate the dashboard

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

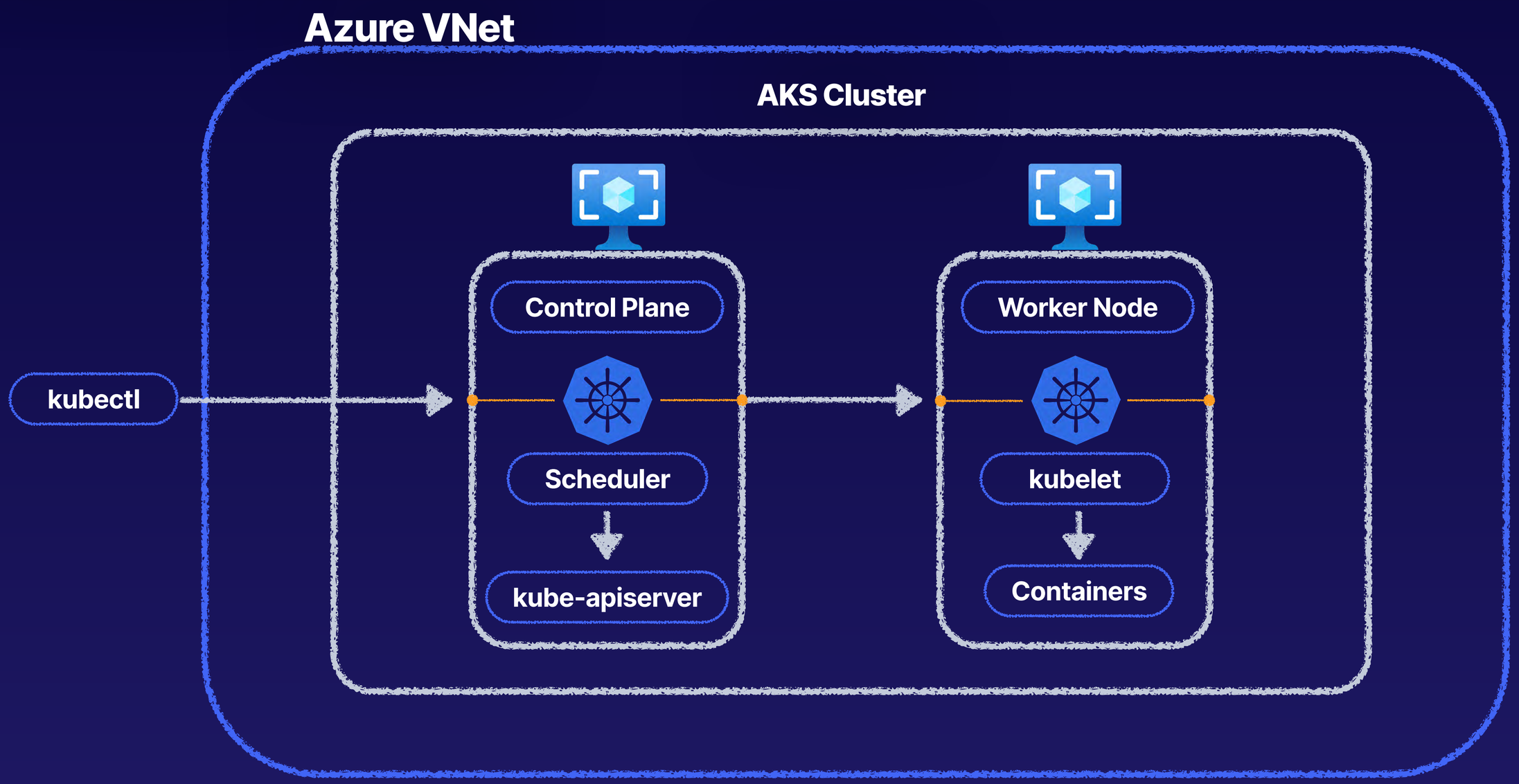
Demo: Provisioning an EKS Cluster

A CLOUD GURU

Demo: Provisioning an AKS Cluster

Jesse Hoch
TRAINING ARCHITECT

Demo Diagram



A CLOUD GURU

Prerequisites for this Demo:

- Install Terraform CLI
- Install kubectl
- Install Azure CLI
- Have ACG's Azure sandbox and credentials
- Have the `demo-terraform-aks-cluster.zip` file

A CLOUD GURU

Demo Objectives:

- Install the Azure CLI
- Install kubectl
- Download the `demo-terraform-aks-cluster.zip` file from the repo
- Unzip the `demo-terraform-aks-cluster.zip` file and then `cd` into the directory
- Create a Service Principal
- Update the `terraform.tfvars` file with the Service Principal

A CLOUD GURU

Demo Objectives:

- Initialize the working directory
- Apply the configuration
- Configure kubectl
- Deploy the Kubernetes dashboard
- Authenticate the dashboard

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Provisioning an AKS Cluster

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Terraform and Kubernetes

SECTION RECAP

Terraform and Kubernetes

Using Terraform to Create and Manage Infrastructure on Kubernetes

Provisioning a Local Kubernetes Cluster

Managing Kubernetes with Terraform

Provisioning an EKS Cluster

Managing EKS with Terraform

Jesse Hoch
Training Architect

A CLOUD GURU

SECTION RECAP

Terraform and Kubernetes

Labs:

- Use Terraform to Create a Kubernetes Deployment
- Manage Kubernetes Resources with Terraform
- Deploy EKS Using Terraform

Jesse Hoch
Training Architect

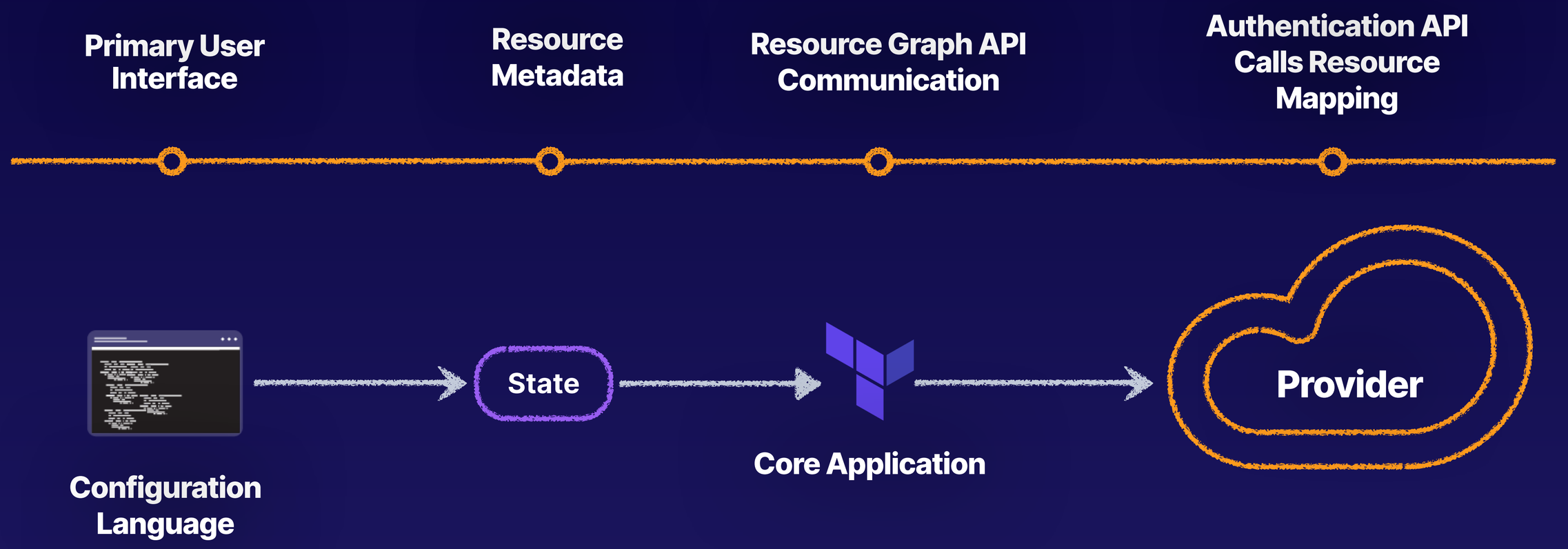
A CLOUD GURU

A CLOUD GURU

Demo: Troubleshooting Terraform

Jesse Hoch
TRAINING ARCHITECT

Demo Diagram



A CLOUD GURU

Prerequisites for this Demo:

- Install Terraform CLI
- Install AWS CLI
- Set up ACG's AWS sandbox
- Have AWS credentials
- Download the `demo-terraform-troubleshooting.zip` file

A CLOUD GURU

Demo Objectives:

- Download the `demo-terraform-troubleshooting.zip` file
- Unzip the file and cd into the directory
- Format the configuration
- Correct a variable interpolation error
- Validate the configuration
- Correct a cycle error
- Correct a `for_each` error

A CLOUD GURU

Demo Objectives:

- Correct the outputs to return all values
- Apply the configuration changes
- Go over bug reporting best practices
- Clean up the workspace

A CLOUD GURU

Jesse Hoch

TRAINING ARCHITECT

Demo: Troubleshooting Terraform

Course Conclusion

Jesse Hoch
Training Architect

A CLOUD GURU

In this course

- ✔ Introduction to Terraform
- ✔ Setting Up Your Environment
- ✔ Terraform Fundamentals
- ✔ Terraform Cloud
- ✔ Using Terraform in a CI/CD Environment
- ✔ Terraform and AWS
- ✔ Terraform and Azure
- ✔ Terraform and Kubernetes
- ✔ Troubleshooting

Jesse Hoch
Training Architect

What’s Next!

- ✔ Deploying to AWS with Terraform and Ansible
- ✔ Managing Microsoft Azure Applications and Infrastructure with Terraform
- ✔ Deploying Resources to GCP with Terraform
- ✔ HashiCorp Certified Terraform Associate