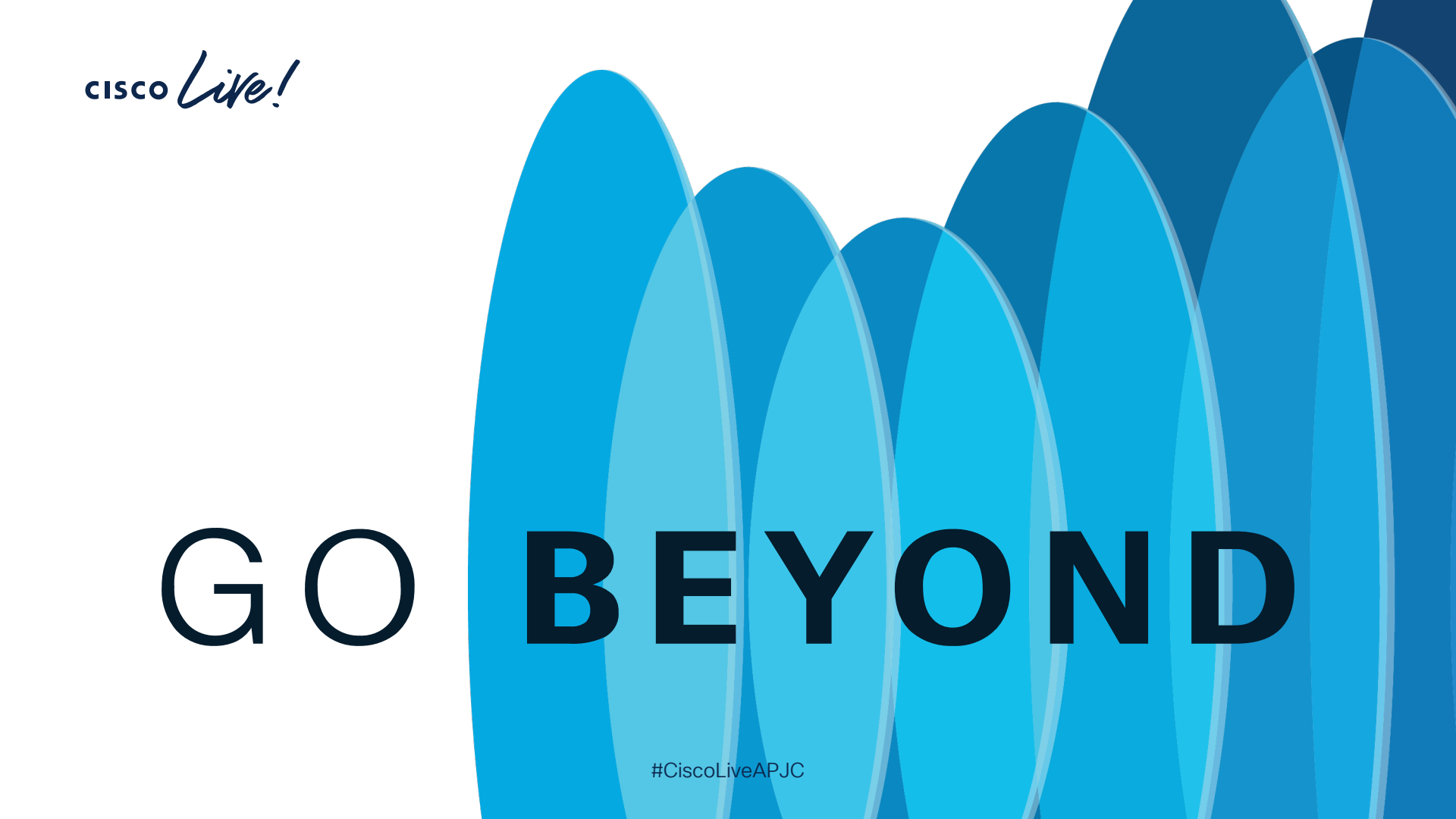

cisco Live!
GO BEYOND
#CiscoLiveAPJC

Advanced Terraform Techniques

Moving Beyond the Scaffolding

Quinn Snyder | Technical Advocate, Cisco Learning and Certifications
@qsnyder

DEVNET-2565

Cisco Webex App

https://ciscolive.ciscoevents.com/ciscolivebot/#DEVNET-2565

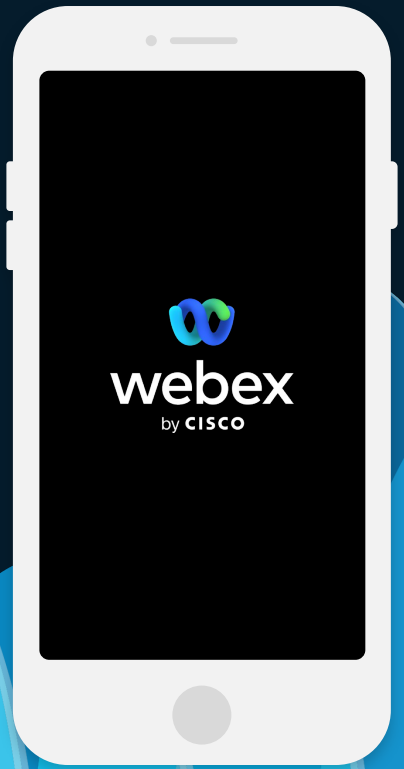
Questions?

Use Cisco Webex App to chat with the speaker after the session

How

1. Find this session in the Cisco Live Mobile App
2. Click “Join the Discussion”
3. Install the Webex App or go directly to the Webex space
4. Enter messages/questions in the Webex space

Webex spaces will be moderated by the speaker until November 15, 2024.

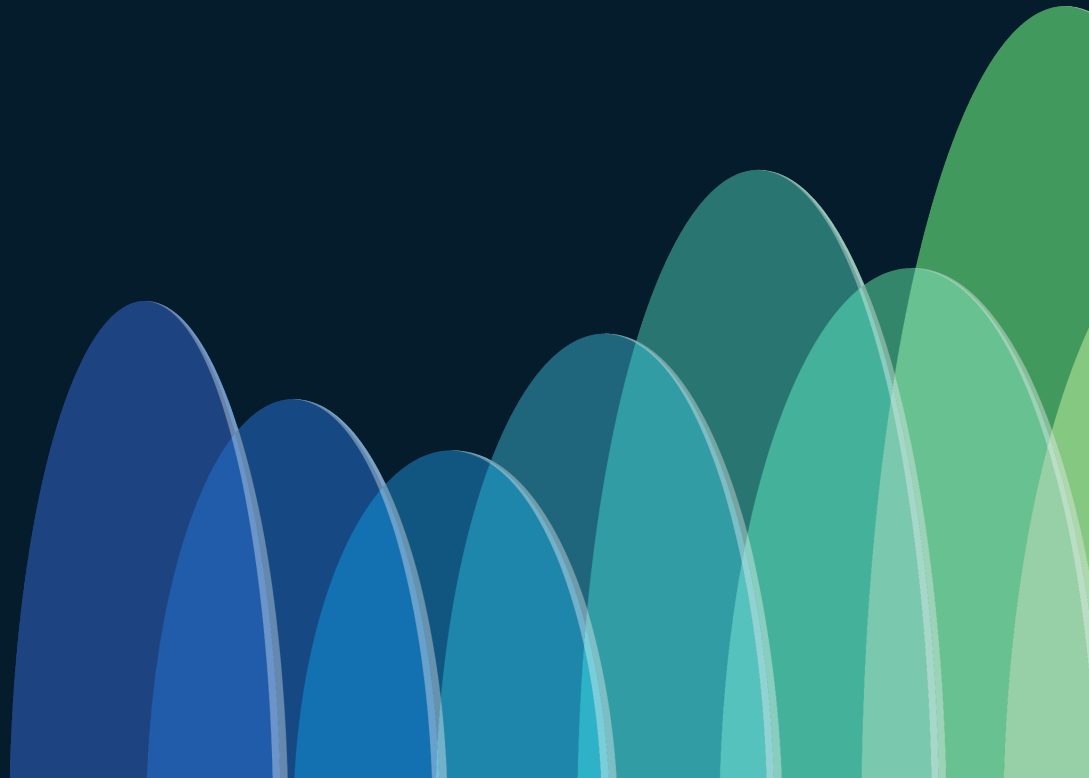


cisco Live!

Agenda

- Introduction
- tfenv
- terraform fmt
- terraform graph
- terraform console
- Terraform modules
- Conclusion

Managing Terraform Versions

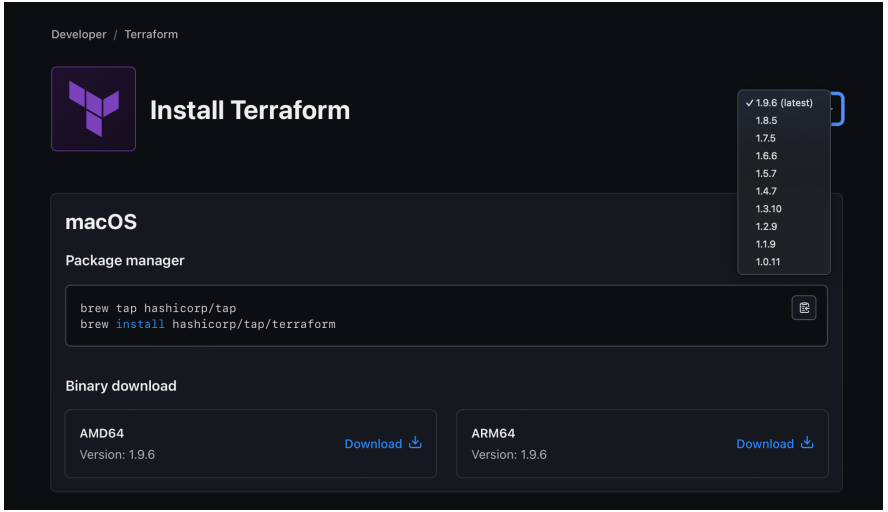


cisco Live!

Terraform installation is a snap...

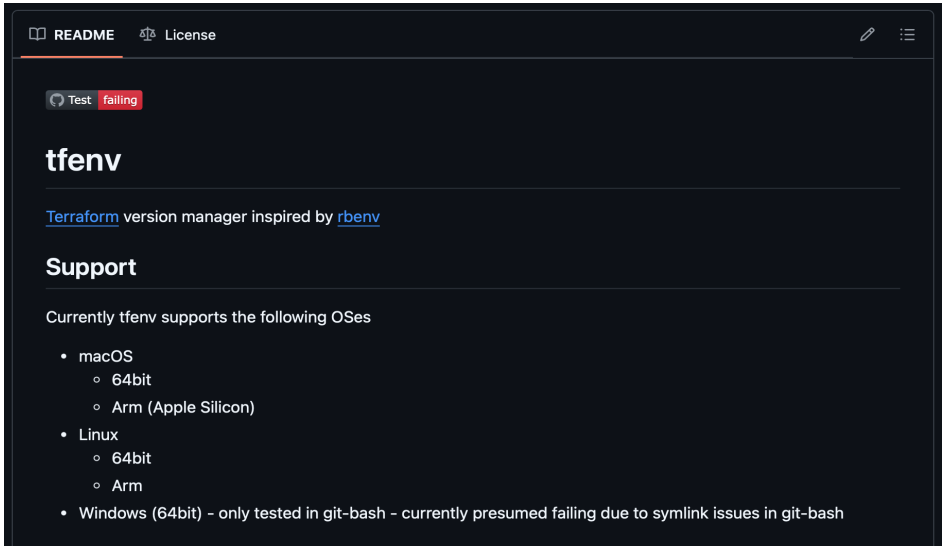
...but how do you handle versioning?

- Terraform uses single, compiled binary (written in Go)
- Support for package managers or direct install to $PATH
- Each binary is still called `terraform`
  - You're on your own for semver



Enter tfenv

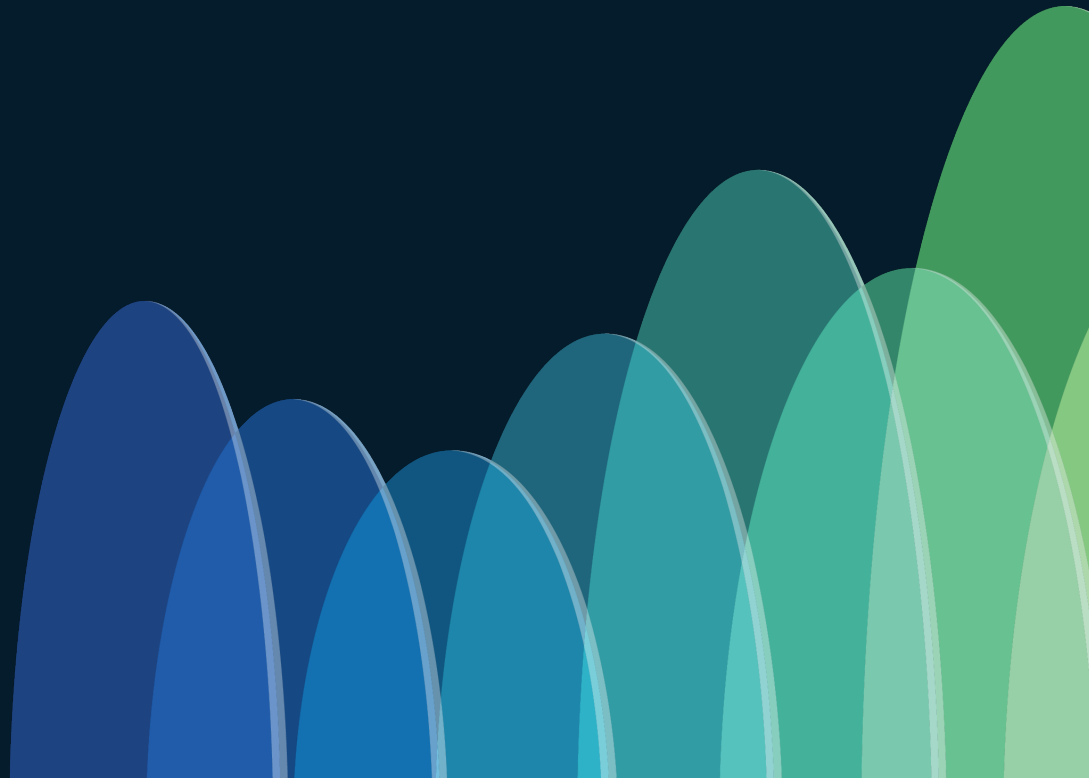
Similar concept to pyenv or rbenv



- CLI tool that manages installed version of terraform in executable path
- Cross-platform support
- This supports all versions of TF
- Can automatically install TF version based on project folder definitions

tfenv Demo

Making your .tf files readable



cisco Live!

Does format and layout matter?

No, but also kinda sorta…

- Terraform is based on Go -> well defined structures and types, along with syntax
- Its not Python (and its supporting cast); whitespace isn't a dealbreaker
- Code can be formatted in such a way to enable easier readability
- Enter `terraform fmt` -> does the heavy-lifting so you don't have to
- Only impacts *.tf files!

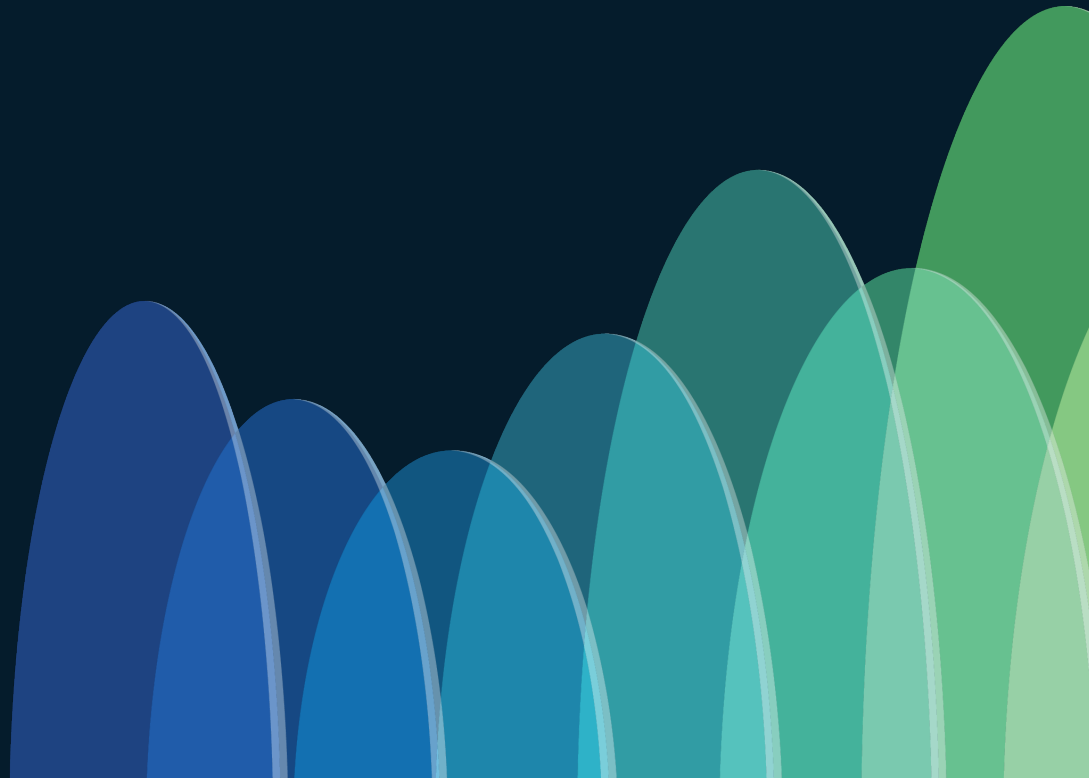
Transform your definitions

```
variable "fabric_inventory" {
description = "Complete inventory for DevNet_Fabric"
type  = map
default = {
spine1 = {
name = "spine1"
ip = "10.10.20.171"
role = "spine"
},
spine2 = {
name = "spine2"
ip = "10.10.20.172"
role = "spine"
},
leaf1 = {
name = "leaf1"
ip = "10.10.20.173"
role = "leaf"
},
leaf2 = {
name = "leaf2"
ip = "10.10.20.174"
role = "leaf"
},
leaf3 = {
name = "leaf3"
ip = "10.10.20.175"
role = "leaf"
},
}
}
```


```
variable "fabric_inventory" {
  description = "Complete inventory for DevNet_Fabric"
  type        = map(any)
  default = {
    spine1 = {
      name = "spine1"
      ip   = "10.10.20.171"
      role = "spine"
    },
    spine2 = {
      name = "spine2"
      ip   = "10.10.20.172"
      role = "spine"
    },
    leaf1 = {
      name = "leaf1"
      ip   = "10.10.20.173"
      role = "leaf"
    },
    leaf2 = {
      name = "leaf2"
      ip   = "10.10.20.174"
      role = "leaf"
    },
    leaf3 = {
      name = "leaf3"
      ip   = "10.10.20.175"
      role = "leaf"
    },
  }
}
```

fmt Demo

Visualizing Object Relationships



cisco Live!

A picture is worth a thousand...

...lines of code?

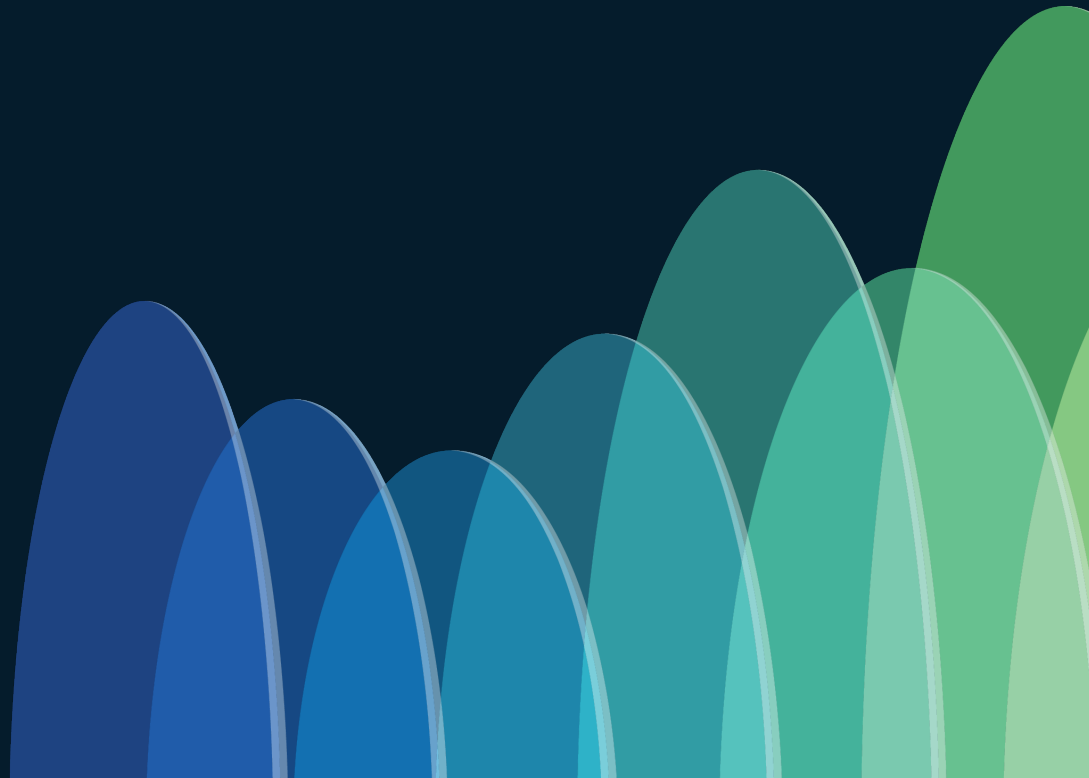
- Terraform is “end-state” declarative; builds resource graph
- This resource graph is generated at all stages; includes state if present
  - Uses all files in project “folder”
- Graph can be visualized using native terraform commands
  - Native output in DOT format (Graphviz)
  - Can be converted to PNG using `dot` command

DOT Syntax

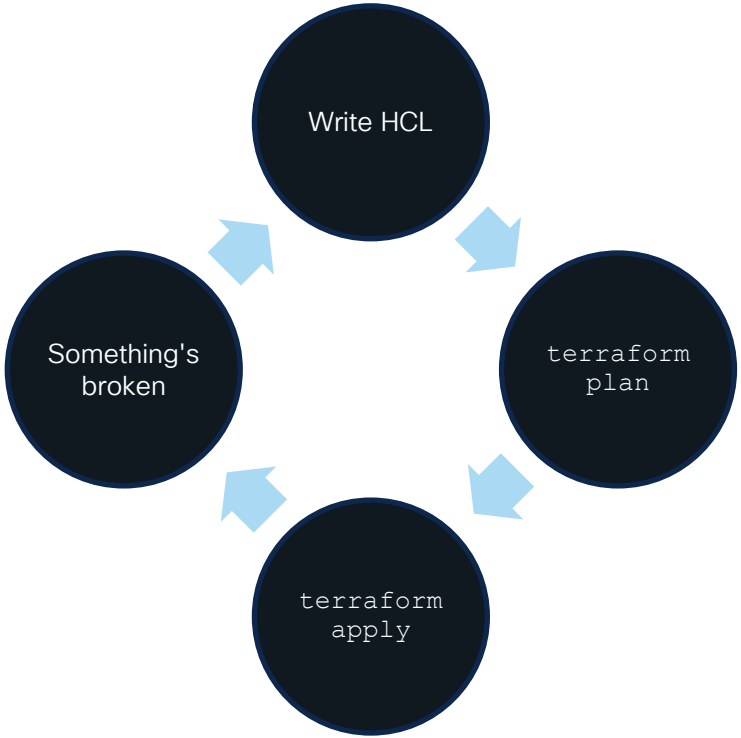
```
digraph {
        compound = "true"
        newrank = "true"
        subgraph "root" {
                "[root] aci_bridge_domain.terraform_bd (expand)" [label = "aci_bridge_domain.terraform_bd", shape =
"box"]
                "[root] aci_subnet.terraform_bd_subnet (expand)" [label = "aci_subnet.terraform_bd_subnet", shape =
"box"]
                "[root] aci_tenant.terraform_tenant (expand)" [label = "aci_tenant.terraform_tenant", shape = "box"]
                "[root] aci_vrf.terraform_vrf (expand)" [label = "aci_vrf.terraform_vrf", shape = "box"]
                "[root] provider[\"registry.terraform.io/ciscodevnet/aci\"]" [label =
"provider[\"registry.terraform.io/ciscodevnet/aci\"]", shape = "diamond"]
                "[root] var.bd" [label = "var.bd", shape = "note"]
                "[root] var.subnet" [label = "var.subnet", shape = "note"]
                "[root] var.tenant" [label = "var.tenant", shape = "note"]
                "[root] var.user" [label = "var.user", shape = "note"]
                "[root] var.vrf" [label = "var.vrf", shape = "note"]
                "[root] aci_bridge_domain.terraform_bd (expand)" -> "[root] aci_vrf.terraform_vrf (expand)"
                "[root] aci_bridge_domain.terraform_bd (expand)" -> "[root] var.bd"
                "[root] aci_subnet.terraform_bd_subnet (expand)" -> "[root] aci_bridge_domain.terraform_bd (expand)"
                "[root] aci_subnet.terraform_bd_subnet (expand)" -> "[root] var.subnet"
                "[root] aci_tenant.terraform_tenant (expand)" -> "[root]
provider[\"registry.terraform.io/ciscodevnet/aci\"]"
                "[root] aci_tenant.terraform_tenant (expand)" -> "[root] var.tenant"
                "[root] aci_vrf.terraform_vrf (expand)" -> "[root] aci_tenant.terraform_tenant (expand)"
                "[root] aci_vrf.terraform_vrf (expand)" -> "[root] var.vrf"
                "[root] provider[\"registry.terraform.io/ciscodevnet/aci\"] (close)" -> "[root]
aci_subnet.terraform_bd_subnet (expand)"
                "[root] provider[\"registry.terraform.io/ciscodevnet/aci\"]" -> "[root] var.user"
                "[root] root" -> "[root] provider[\"registry.terraform.io/ciscodevnet/aci\"] (close)"
        }
}
```

graph Demo

Interactive(ish) Terraform



How do we interact with Terraform plans?



How do we interact with Terraform plans?

- This works fine for simple HCL, since errors are probably typos or syntactical
- Single level of variable creation (maps, assignments) make for easy troubleshooting

How do we interact with Terraform to ensure some level of “function” prior to performing the TF cycle?

Terraform has a console!

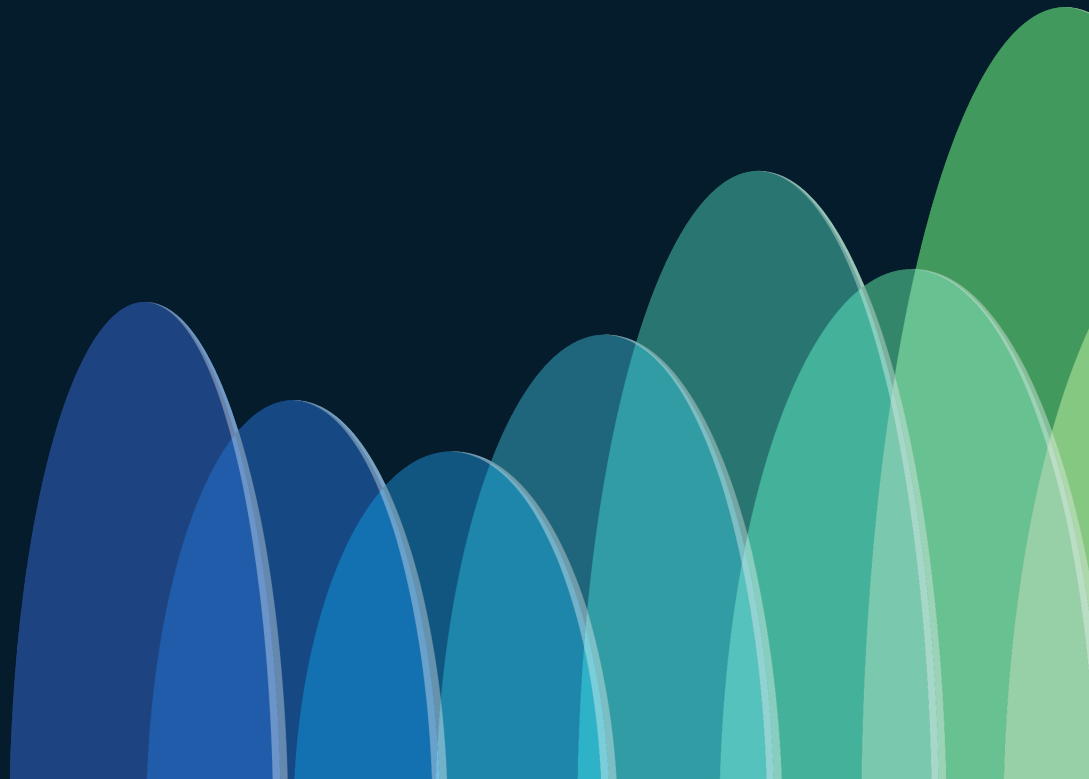
- `terraform console` allows for interactive interrogation of Terraform
  - Variables
  - Configurations
  - State (if present)

- Perform variable and data interpolations based on built-in functions within Terraform

However, there is no Terraform REPL!

console Demo

Simplifying the Automation

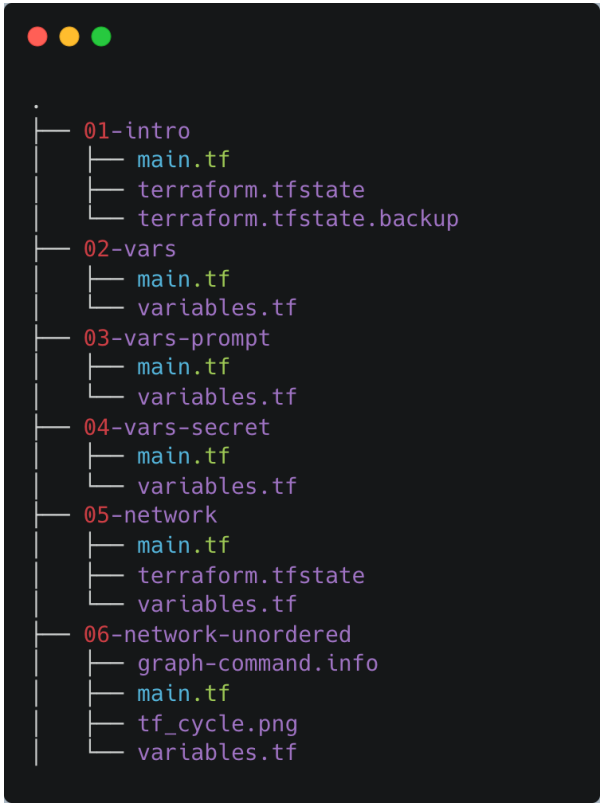


cisco Live!

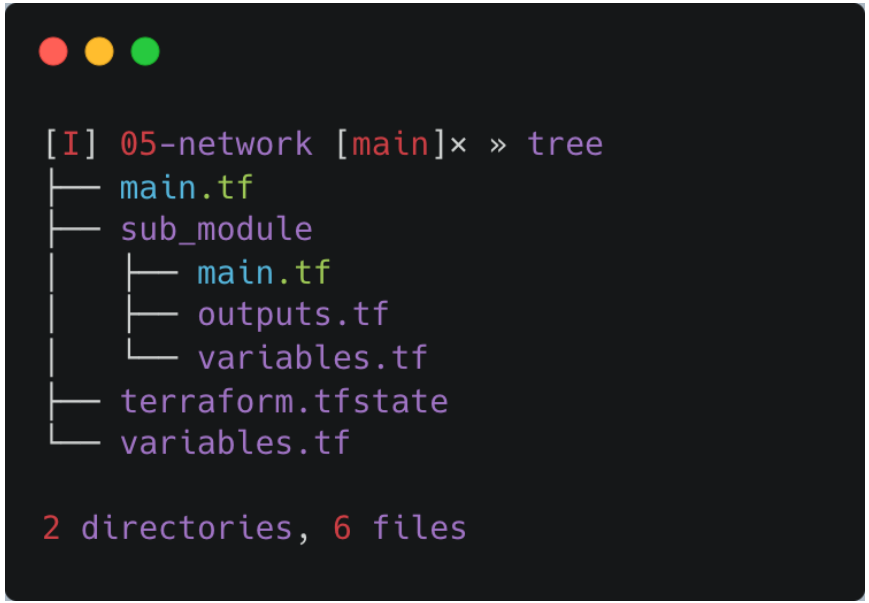
Ansible has roles, what about Terraform?

Terraform uses the concept of “modules” for simplification and abstraction of a configuration

Every “project directory” is a “module” in the eyes of Terraform



Ansible has roles, what about Terraform?



Recall within a root “module”, all .tf files are analyzed an interpolated; even in subfolders

This builds the concept for a module:

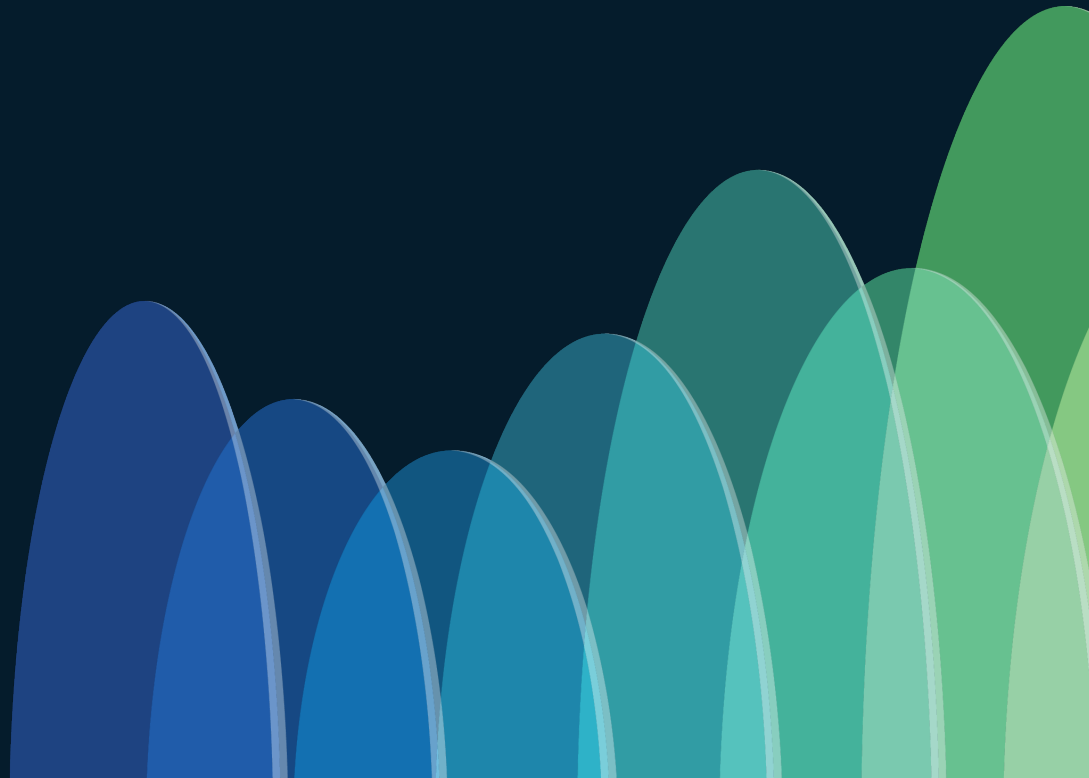
- Defined structure of desired state
- Abstract underlying complexity
- Feed only required info

When to build modules?

- Simplify configuration
  - Abstract the complexity and present only top-level variables and resources
- Prevent configuration issues
  - Removal of configuration options provides easier to interrogate automation
- Configuration re-use
  - Common sets of automation routines can be packaged and shared across teams/organizations

Module Demo

Wrapping it All Up



cisco Live!

Sample Code Is Here!

Complete Your Session Evaluations

Complete a minimum of 4 session surveys and the Overall Event Survey to claim a **Cisco Live T-Shirt**.

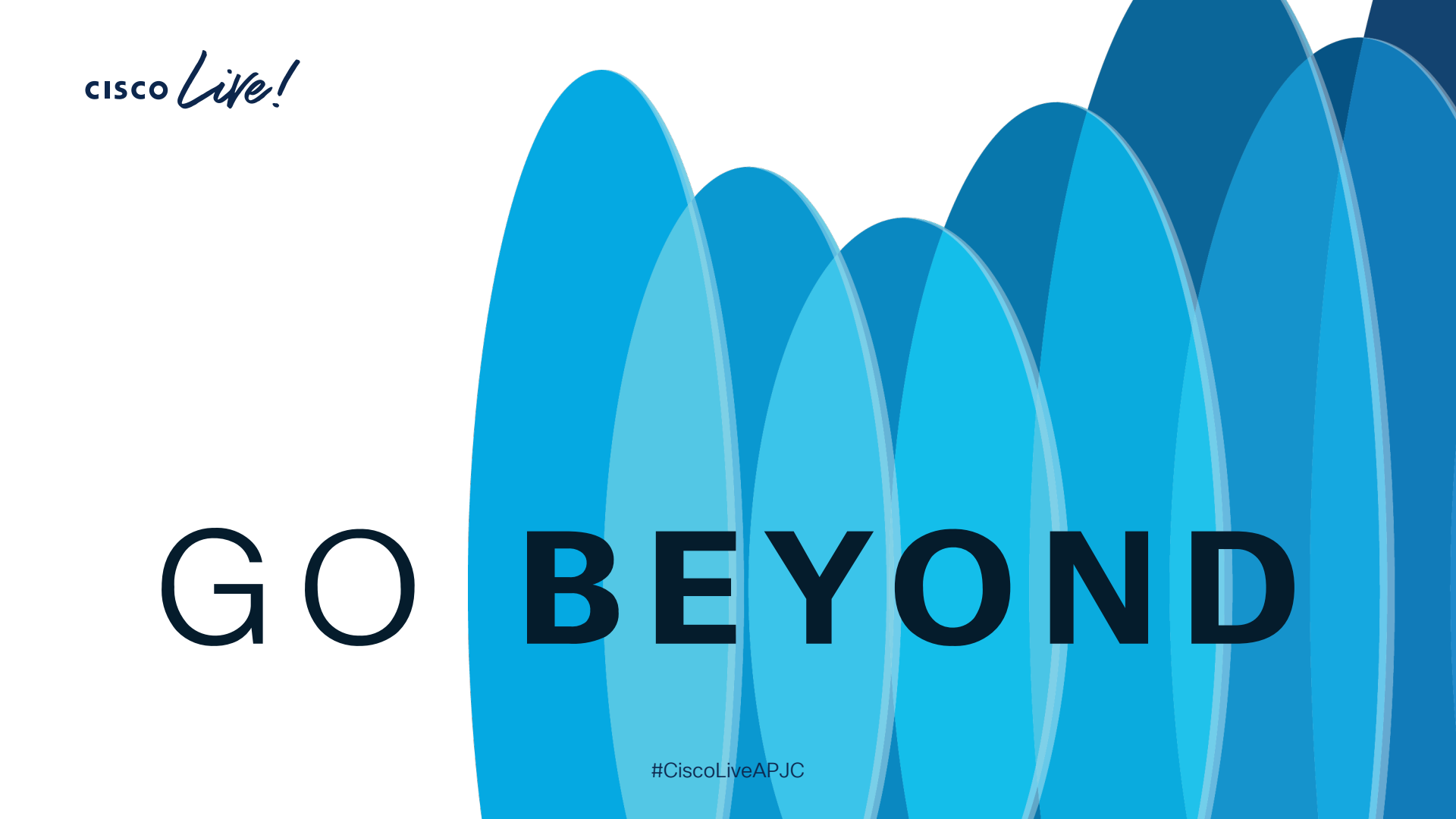
Complete your surveys in the **Cisco Live mobile app**.

Continue your education

- Visit the Cisco Stand for related demos
- Book your one-on-one Meet the Expert meeting
- Attend the interactive education with DevNet, Capture the Flag, and Walk-in Labs
- Visit the On-Demand Library for more sessions at www.CiscoLive.com/on-demand

Contact me at: Insert preferred comms method


cisco
Thank you
cisco Live!
#CiscoLiveAPJC


cisco Live!
GO BEYOND
#CiscoLiveAPJC