The bridge to possible

Introduction to Infrastructure as Code for ACI with Terraform

Thomas Renzy, Technical Leader CX
@ThomasRenzy

BRKDCN-2607

Cisco Webex App

Questions?

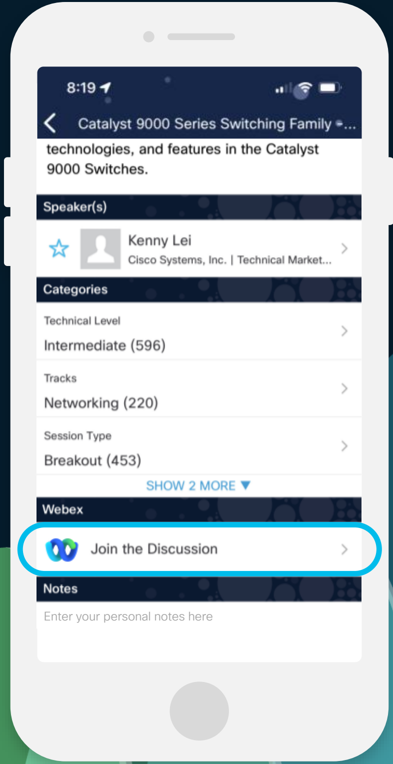
Use Cisco Webex App to chat
with the speaker after the session

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 June 7, 2024.

https://ciscolive.ciscoevents.com/
ciscolivebot/#BRKDCN-2607



8:19

Catalyst 9000 Series Switching Family ...

technologies, and features in the Catalyst 9000 Switches.

Speaker(s)

Kenny Lei
Cisco Systems, Inc. | Technical Market...

Categories

Technical Level
Intermediate (596)

Tracks
Networking (220)

Session Type
Breakout (453)

SHOW 2 MORE ▼

Webex

Join the Discussion

Notes

Enter your personal notes here

Agenda

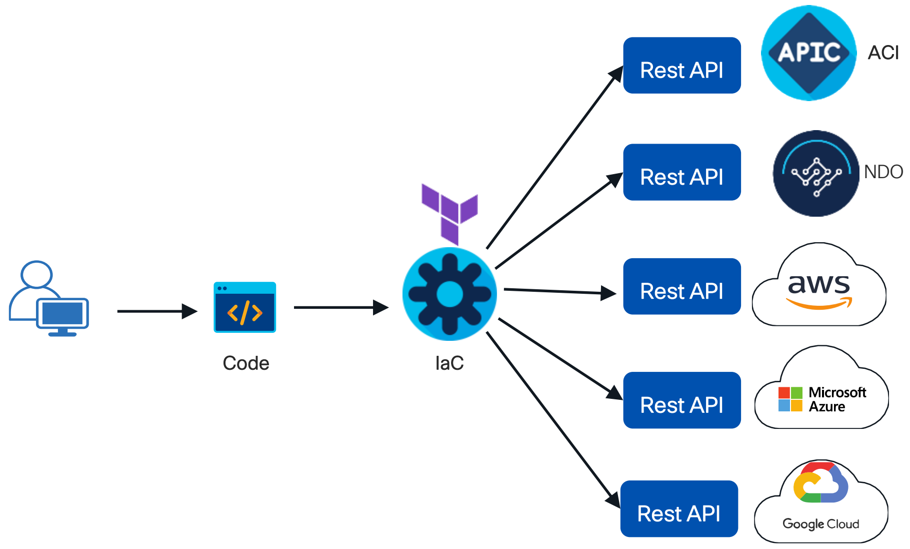
- What is Infrastructure as Code?
- Infrastructure as Code with Terraform
- Terraform Providers
- Resources & Data Sources
- Deploying IaC with Terraform
- Key Takeaways

What is Infrastructure as Code?

(Why should we care?)

What is Infrastructure as Code(IaC)?

The management & provisioning of computer infrastructure through code and data structures instead of direct device management.



Benefits of Infrastructure as Code(IaC)?

- Provisioning/repeatable tasks
- Provides Speed & Scale
- Cost savings
- Reduce Errors
- Improve infrastructure consistency

“Isn’t that for Cloud Infrastructure???

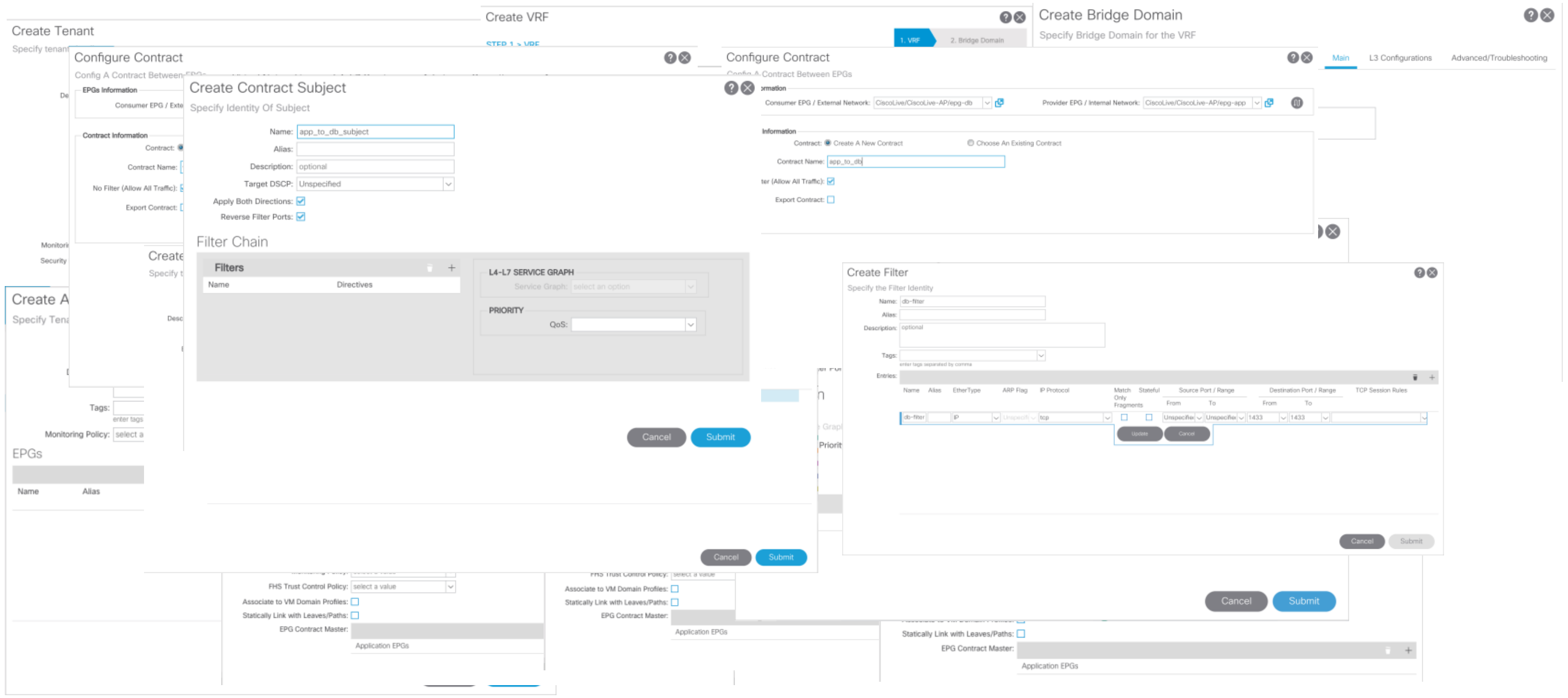
Not Anymore - ACI & NDO Robust REST API

”I’m not a programmer!"

Don’t have to be

IaC is another tool in our toolbox

“ClickOps”– APIC GUI



Create Tenant
Specify tenant

Create VRF
STEP 1 > VRF
1. VRF 2. Bridge Domain

Create Bridge Domain
Specify Bridge Domain for the VRF
Main L3 Configurations Advanced/Troubleshooting

Configure Contract
Config A Contract Between EPGs
EPGs Information
Consumer EPG / External Network: CiscoLive/CiscoLive-AP/epg-db
Provider EPG / Internal Network: CiscoLive/CiscoLive-AP/epg-app
Contract Information
Contract: ● Create A New Contract ○ Choose An Existing Contract
Contract Name: app_to_db
No Filter (Allow All Traffic): ✓
Export Contract: ☐

Create Contract Subject
Specify Identity Of Subject
Name: app_to_db_subject
Alias:
Description: optional
Target DSCP: Unspecified
Apply Both Directions: ✓
Reverse Filter Ports: ✓

Filter Chain

Name	Directives

L4-L7 SERVICE GRAPH
Service Graph: select an option
PRIORITY
QoS:

Cancel Submit

Create Filter
Specify the Filter Identity
Name: db-filter
Alias:
Description: optional
Tags:
enter tags separated by comma
Entries:

Name	Alias	EtherType	ARP Flag	IP Protocol	Match Only Fragments	Stateful	Source Port / Range From	Source Port / Range To	Destination Port / Range From	Destination Port / Range To	TCP Session Rules
db-filter		IP	Unspecified	tcp	☐	☐	Unspecified	Unspecified	1433	1433	

Update Cancel

Cancel Submit

Create Application EPG
Specify Tenant
Tags: enter tags
Monitoring Policy: select a
EPGs

Name	Alias

FHS Trust Control Policy: select a value
Associate to VM Domain Profiles: ☐
Statically Link with Leaves/Paths: ☐
EPG Contract Master:
Application EPGs

Cancel Submit

Infrastructure as Code with Terraform

What is Terraform?

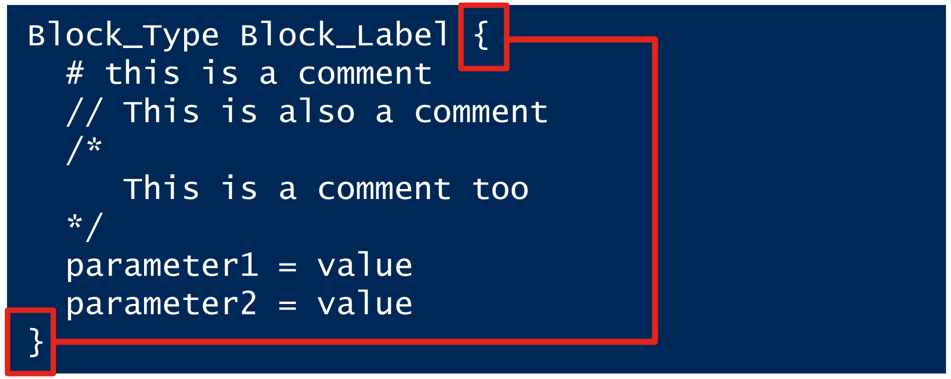
HashiCorp Terraform

- Open Source
- Infrastructure Provisioning
- Single Binary (Windows, Mac, Linux, Solaris, FreeBSD)
- Declarative and Stateful
- HashiCorp Configuration Language (HCL)
- APIC/NDO REST API Interaction
- No Special Programming Skills required

HashiCorp Configuration Language (HCL)

- Based on HCL2 toolkit
  - Declarative Language - HashiCorp
  - Primary user interface
- Designed for Infrastructure Provisioning

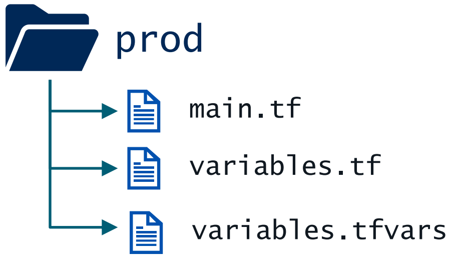
- Built around
  - Blocks – Resources/Data Sources/Modules
  - Parameters/Arguments
  - Blocks open "{" and "}"



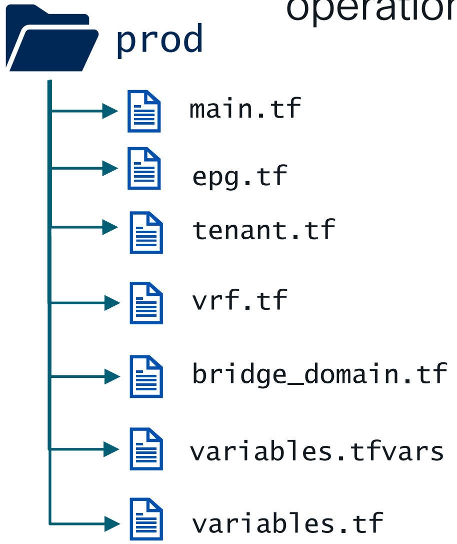
```
Block_Type Block_Label {
  # this is a comment
  // This is also a comment
  /*
     This is a comment too
  */
  parameter1 = value
  parameter2 = value
}
```

Terraform Plans/Configuration Files

- Collection of HCL instructions
  - What you want to provision (intent)
  - *.tf extension
- Terraform scans directory
  - Directory that terraform is run in
  - Declarative – No need for order of operations



or



Terraform – Single Binary

- Reading .tf & .tfvars files
- State Management
- Building Graph
- Plan execution

```
> terraform -help
Usage: terraform [global options] <subcommand> [args]

The available commands for execution are listed below.
The primary workflow commands are given first, followed by
less common or more advanced commands.

Main commands:
  init          Prepare your working directory for other commands
  validate      Check whether the configuration is valid
  plan          Show changes required by the current configuration
  apply         Create or update infrastructure
  destroy       Destroy previously-created infrastructure
```

Terraform Block Configuration

- Configures Terraform behavior
- Version of Terraform binary
- Specify the Providers you want

```
terraform {
  required_version ">= 1.6.0"
  required_providers {
    aci = {
      source = "CiscoDevNet/aci”
      version = “2.11.1”
    }
    mso = {
      source = "CiscoDevNet/mso”
  }
}
```

Specify the version of the Binary (version constraint)

Provider information

Terraform Providers

Terraform Providers

- Terraform Binary doesn't know ACI/NDO
- Providers Understand API interactions
  - APIC and MSO REST API calls
- Relies on specific plugins
  - Installed via `terraform init`

Official	Maintained by HashiCorp Ex. AWS, Azure, GCP
Partner	Maintained by partners Ex. ACI, MSO, ASA
Community	Open-Source Community

Terraform Provider configuration (ACI)

BU & TAC Support

```
terraform {
  required_version ">= 1.6.0"
  required_providers {
    aci = {
      source = "CiscoDevNet/aci”
      version = “2.13.0”
    }
  }
}
```

**Can specify Multiple Providers

terraform init

- Download and Installs plugins
- Must initialize before plan/apply
- Creates a provider “lock” file
- Can specify version

Provider Block (ACI)

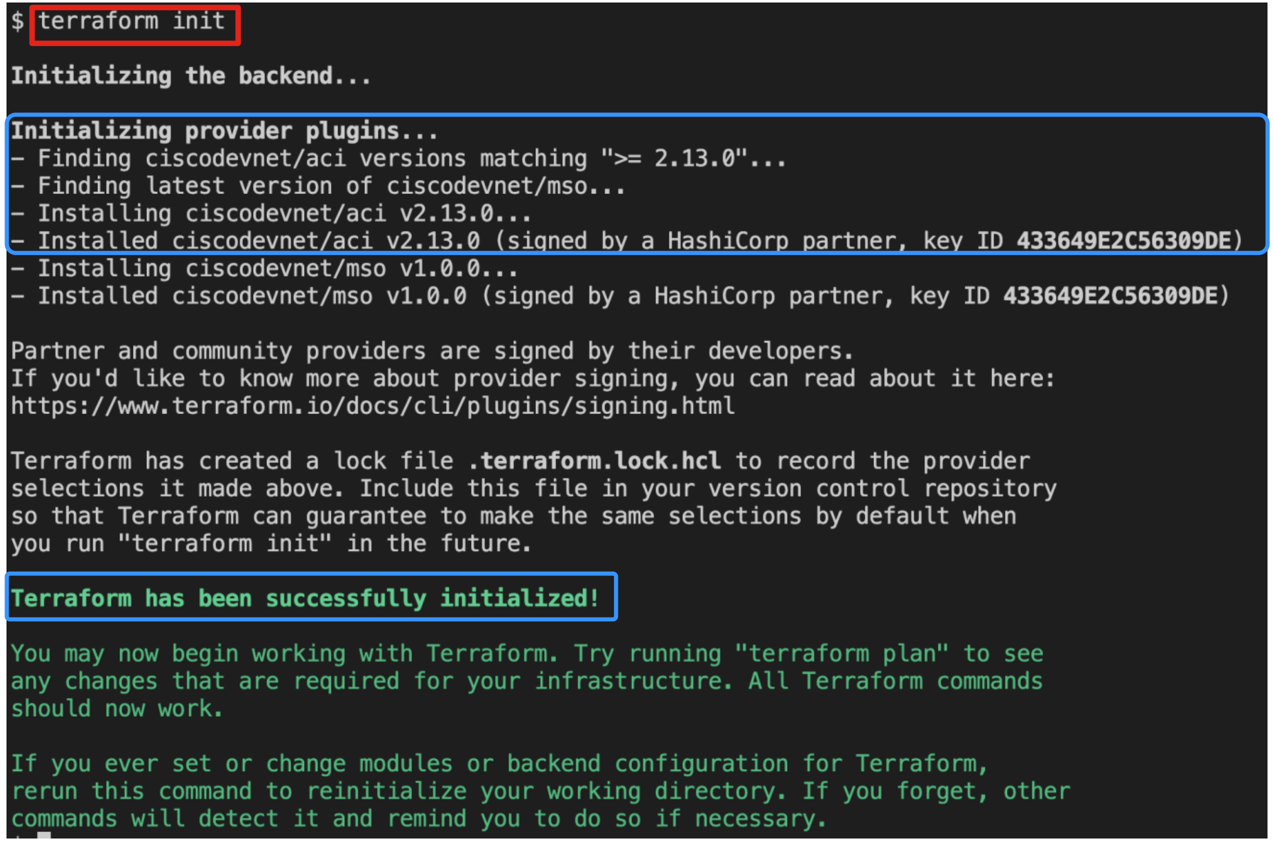
```
provider "aci" {
  # cisco-aci user name
  username = “admin
  # Password
  password = ”cisco.123"
  # cisco-aci url
  url      = "https://172.31.2.31/"
  insecure = true
}
```

Authentication

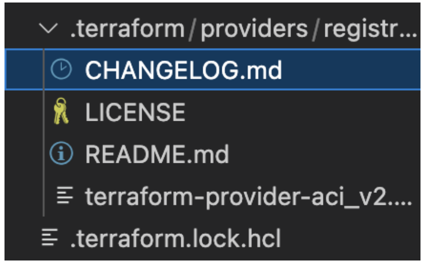
APIC URL

No SSL Cert Validation

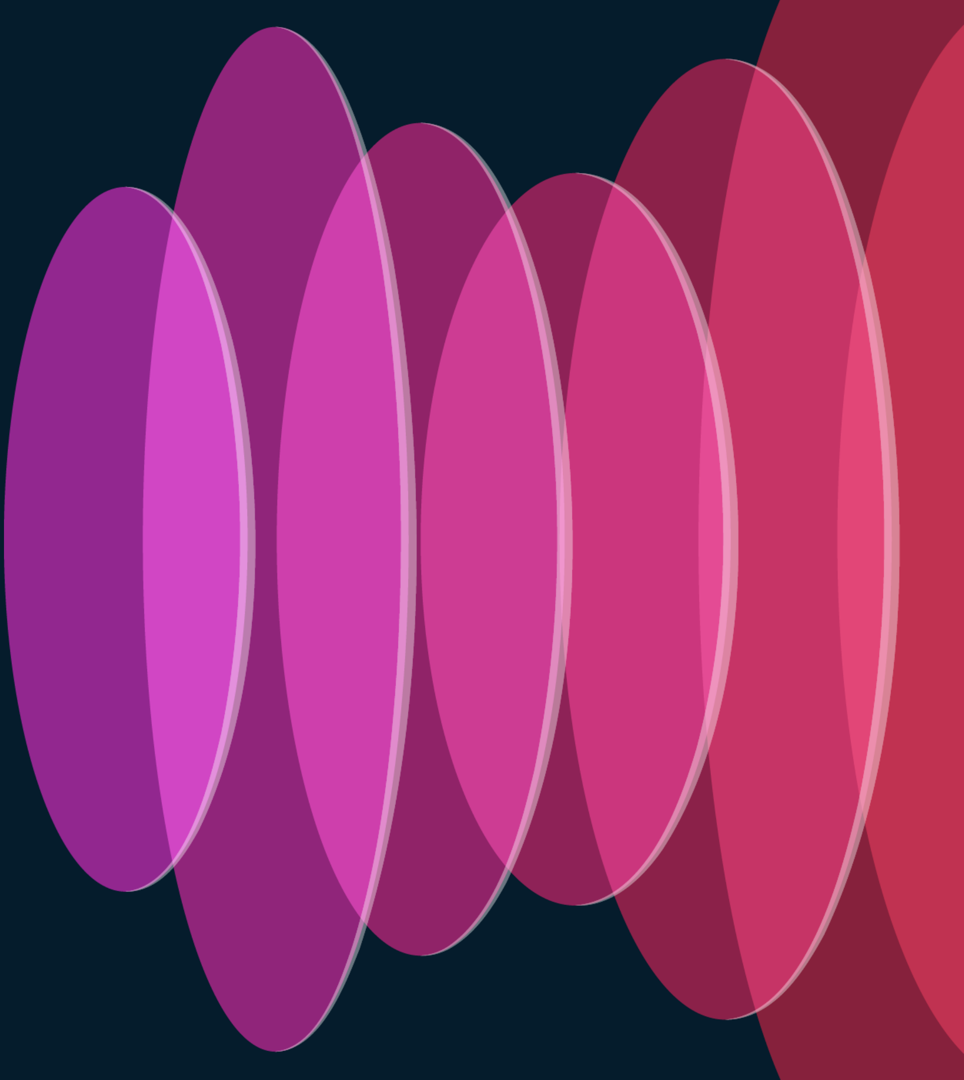
Terraform init



```
$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding ciscodevnet/aci versions matching ">= 2.13.0"...
- Finding latest version of ciscodevnet/mso...
- Installing ciscodevnet/aci v2.13.0...
- Installed ciscodevnet/aci v2.13.0 (signed by a HashiCorp partner, key ID 433649E2C56309DE)
- Installing ciscodevnet/mso v1.0.0...
- Installed ciscodevnet/mso v1.0.0 (signed by a HashiCorp partner, key ID 433649E2C56309DE)

Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/cli/plugins/signing.html

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

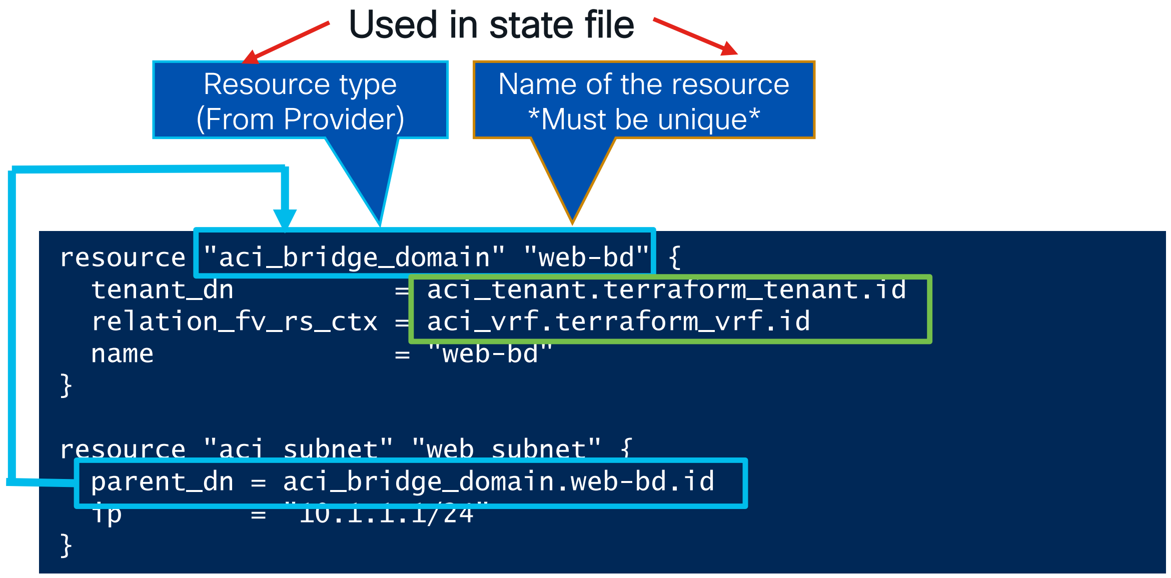

.terraform / providers / registr...
CHANGELOG.md
LICENSE
README.md
terraform-provider-aci_v2....
.terraform.lock.hcl

Terraform Resources & Data Sources



Terraform Resources - ACI

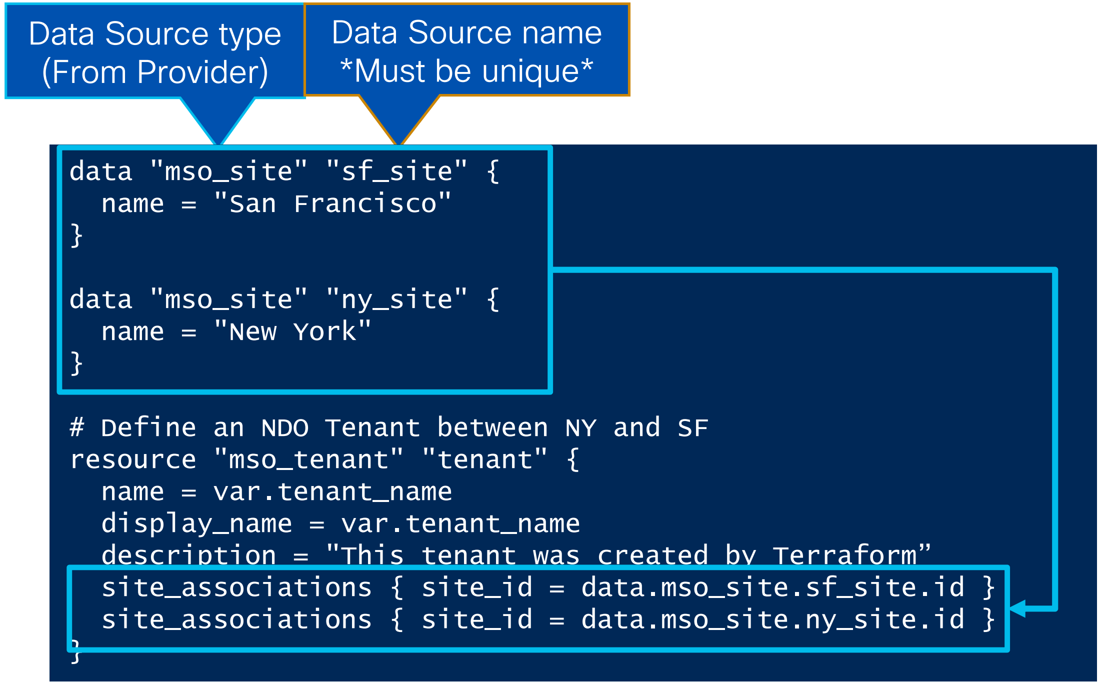
- Represents Infrastructure to be managed
  - Defines desired state
- Accepts arguments
  - Required
  - Optional


Used in state file
Resource type (From Provider)
Name of the resource *Must be unique*

```
resource "aci_bridge_domain" "web-bd" {
  tenant_dn                 = aci_tenant.terraform_tenant.id
  relation_fv_rs_ctx = aci_vrf.terraform_vrf.id
  name                          = "web-bd"
}

resource "aci_subnet" "web_subnet" {
  parent_dn = aci_bridge_domain.web-bd.id
  ip             = "10.1.1.1/24"
}
```

Terraform Data Sources - ACI

- Allows data to be fetched for use elsewhere in Terraform configuration


Data Source type (From Provider)
Data Source name *Must be unique*

```
data "mso_site" "sf_site" {
  name = "San Francisco"
}

data "mso_site" "ny_site" {
  name = "New York"
}

# Define an NDO Tenant between NY and SF
resource "mso_tenant" "tenant" {
  name = var.tenant_name
  display_name = var.tenant_name
  description = "This tenant was created by Terraform”
  site_associations { site_id = data.mso_site.sf_site.id }
  site_associations { site_id = data.mso_site.ny_site.id }
}
```

aci_rest_managed

- When there isn’t a Resource
- Manages Objects via REST API calls with no resource
- Can reconcile state information

```
resource "aci_rest_managed" ”rest_tenant" {
  dn         = "uni/tn-REST"
  class_name = "fvTenant"
  content = {
    name   = ”REST"
    descr = ”Tenant built with REST"
  }
}
```

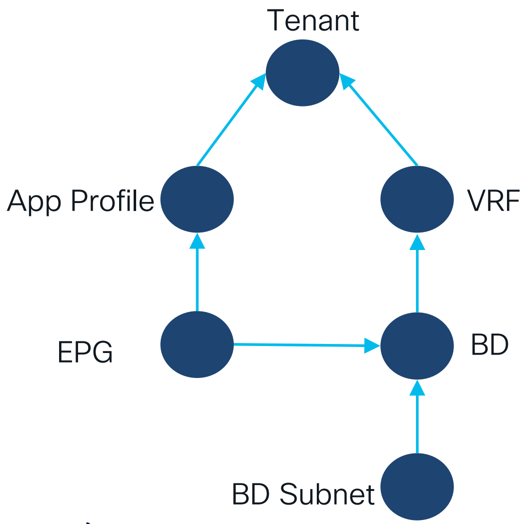
Terraform State

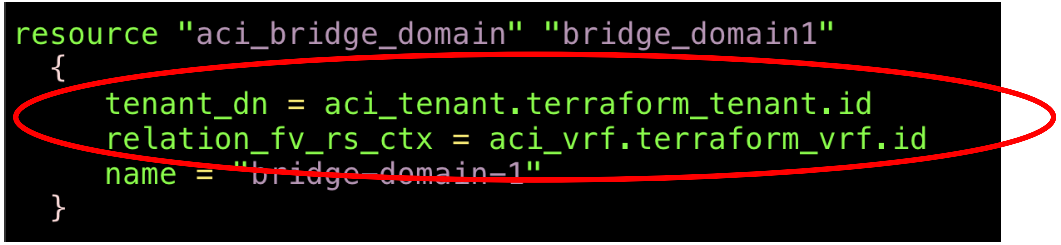
- Terraform is stateful
  - Tracks objects it builds (terraform.tfstate)
  - Source of everything it knows about
- Stored inside working directory
  - Can use backend – AWS, Terraform Cloud
  - <u>Do not modify state file directly</u>

```
{
  "version": 4,
  "terraform_version": "1.7.0",
  "serial": 18,
  "lineage": "f7aa5662-7643-c475-830e-a76433b16ef2",
  "outputs": {},
  "resources": [
    {
      "mode": "managed",
      "type": "aci_bridge_domain",
      "name": "web-bd",
      "provider": "provider[\"registry.terraform.io/ciscodevnet/aci\"]",
      "instances": [
        {
          "schema_version": 1,
          "attributes": {
            "annotation": "orchestrator:terraform",
            "description": "Created by Terraform",
```

Terraform Dependency Mapping

- Uses Graphs to track of dependencies and correct order of deployment
- Builds a graph of relationships (Directional tree without loops)

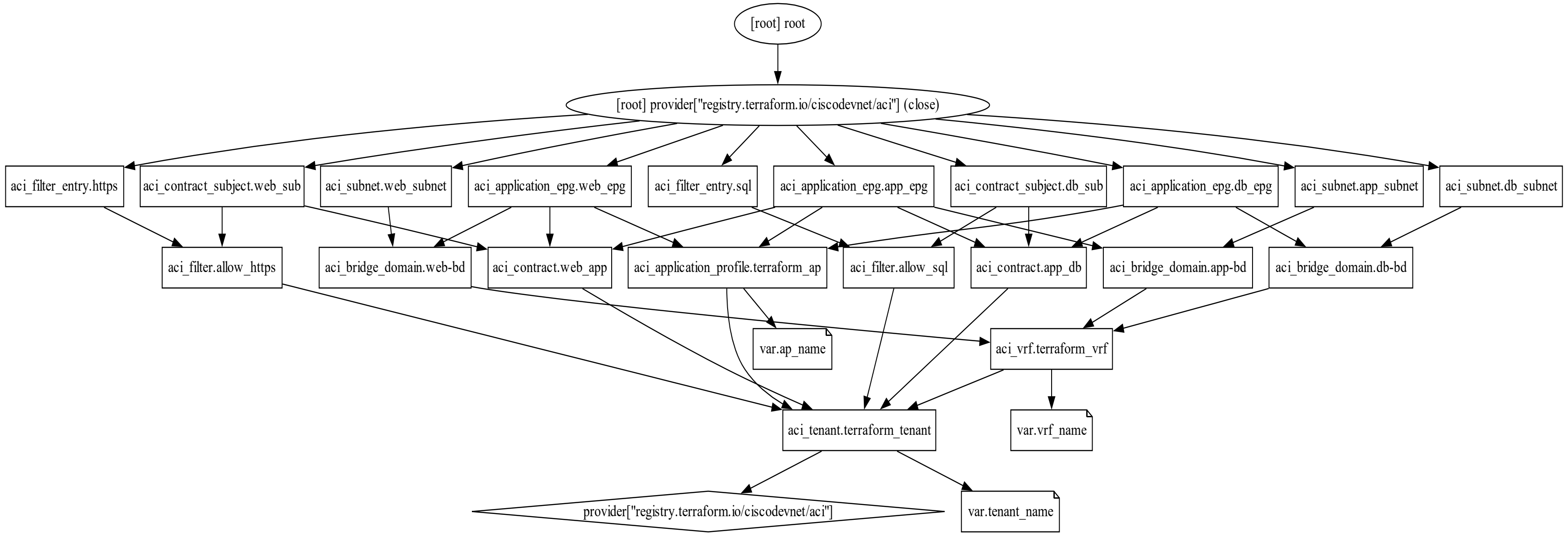


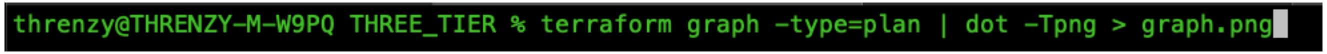


```
resource "aci_bridge_domain" "bridge_domain1"
  {
     tenant_dn = aci_tenant.terraform_tenant.id
     relation_fv_rs_ctx = aci_vrf.terraform_vrf.id
     name = "bridge-domain-1"
  }
```

Reference Slide

Terraform Graph (Three Tier)





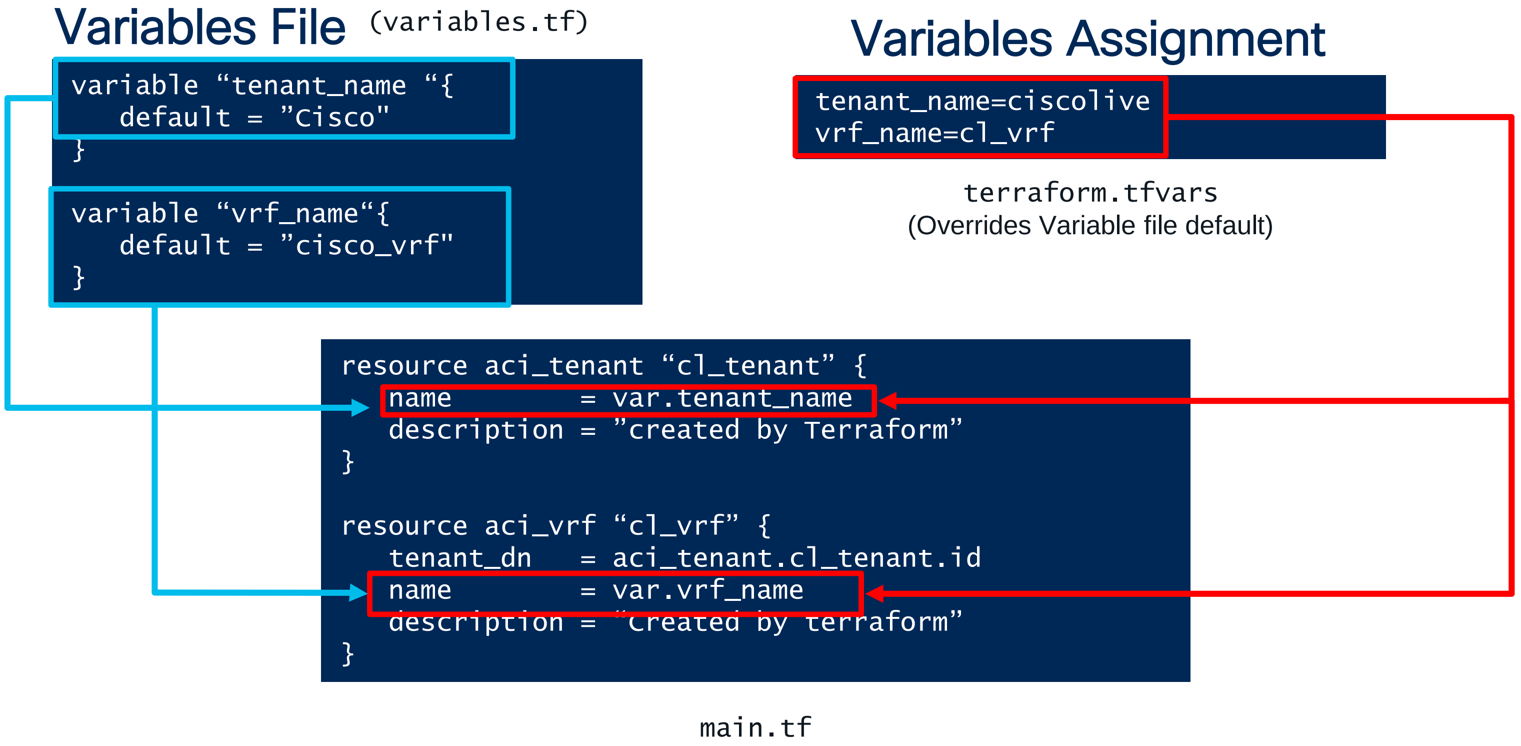
```
threnzy@THRENZY-M-W9PQ THREE_TIER % terraform graph -type=plan | dot -Tpng > graph.png
```

Terraform Variables & Iteration

Variables in Terraform

- Makes code reusable
  - Values no longer hard-coded
- Defined in a separate file (variables.tf & *.tfvars)
  - Separate data from logic
- Variables have
  - Type – string, number, bool, list, set, map, object
  - Default value
    - If no value is set, user will be prompted for value
  - Description

Terraform Variables



Variables File (variables.tf)

```
variable “tenant_name “{
   default = ”Cisco"
}

variable “vrf_name“{
   default = ”cisco_vrf"
}
```

Variables Assignment

```
tenant_name=ciscolive
vrf_name=cl_vrf
```

terraform.tfvars
(Overrides Variable file default)

```
resource aci_tenant “cl_tenant” {
   name        = var.tenant_name
   description = ”created by Terraform”
}

resource aci_vrf “cl_vrf” {
   tenant_dn   = aci_tenant.cl_tenant.id
   name        = var.vrf_name
   description = “created by terraform”
}
```

main.tf

Reference Slide

Terraform Variables Precedence

- Variables have precedence
- Variables can be set, but overridden

Higher ↑

- Command Line Flag run as command line switch
- Configuration file - set in your terraform.tfvars file
- Environment variable - part of your shell environment
- Default Config - default value in variables.tf
- User manual entry - if not specified, prompt the user for entry

lower ↓

https://developer.hashicorp.com/terraform/language/values/variables

Iteration (loop) in Terraform - count

```
resource "aci_bridge_domain" "count_bd_1" {
  tenant_dn    = aci_tenant.count_tenant.id
  relation_fv_rs_ctx = aci_vrf.terraform_vrf.id
  description = "Created with Terraform count"
  name        = bd_1
  arp_flood   = "yes"
}
```

```
resource "aci_bridge_domain" "count_bd_2" {
  tenant_dn    = aci_tenant.count_tenant.id
  relation_fv_rs_ctx = aci_vrf.terraform_vrf.id
  description = "Created with Terraform count"
  name        = bd_2
  arp_flood   = "yes"
}
```

```
resource "aci_bridge_domain" "count_bd_3" {
  tenant_dn    = aci_tenant.count_tenant.id
  relation_fv_rs_ctx = aci_vrf.terraform_vrf.id
  description = "Created with Terraform count"
  name        = bd_3
  arp_flood   = "yes"
}
```

- count
  - Add number of resources based on count
  - bd_1, bd_2, bd_3

```
resource "aci_bridge_domain" "count_bd" {
  count         = 3
  tenant_dn     = aci_tenant.count_tenant.id
  relation_fv_rs_ctx = aci_vrf.terraform_vrf.id
  description = "Created with Terraform count"
  name          = "bd_${count.index}"
  arp_flood     = "yes"
}
```

Iteration (loop) in Terraform – for_each

- `for_each`
  - Create objects based on a set or map

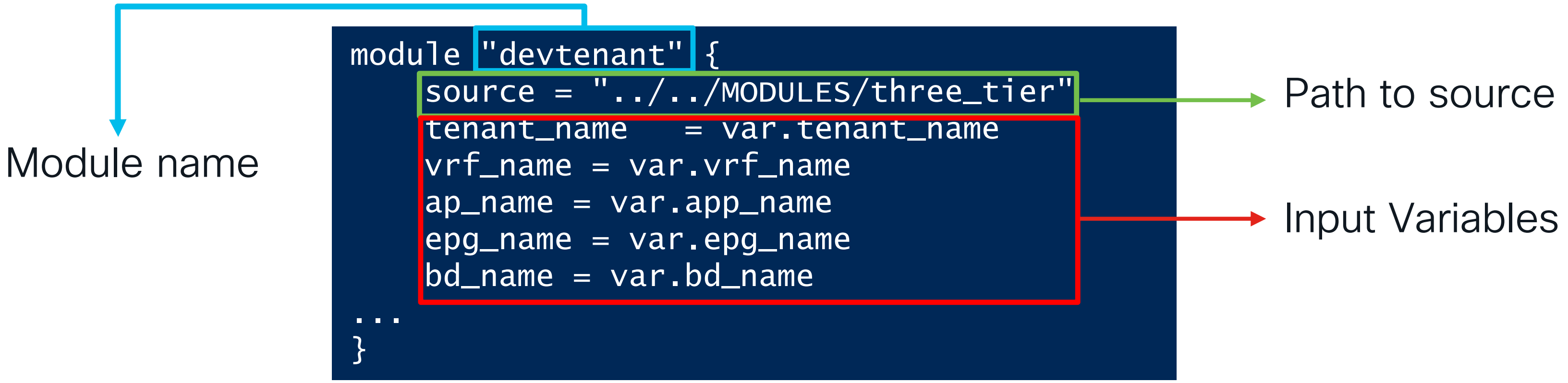
```
resource "aci_bridge_domain" ”prod" {
  tenant_dn    = aci_tenant.count_tenant.id
  relation_fv_rs_ctx = aci_vrf.terraform_vrf.id
  description = "Created with Terraform count"
  name         = ”prod"
  arp_flood    = "yes"
}
```

```
resource "aci_bridge_domain" ”dev" {
  tenant_dn    = aci_tenant.count_tenant.id
  relation_fv_rs_ctx = aci_vrf.terraform_vrf.id
  description = "Created with Terraform count"
  name         = ”dev"
  arp_flood    = "yes"
}
```

```
resource "aci_bridge_domain" ”each_bd" {
  for_each = var.bds
  tenant_dn    = aci_tenant.count_tenant.id
  relation_fv_rs_ctx = aci_vrf.terraform_vrf.id
  description = "Created with Terraform"
  name         = each.value.name
  arp_flood    = "yes"
}
```

Terraform Modules – Code Reusability

- Modules create reusable components
  - DRY – Don’t repeat yourself
  - Like a function in programming languages
- Modules take inputs and (optionally) return outputs
- Modules are made of resources or other modules


```
module "devtenant" {
    source = "../../MODULES/three_tier"
    tenant_name    = var.tenant_name
    vrf_name = var.vrf_name
    ap_name = var.app_name
    epg_name = var.epg_name
    bd_name = var.bd_name
...
}
```

Terraform Modules

"../../MODULES/three_tier

```
terraform {
  required_providers {
    aci = {
      source  = "CiscoDevNet/aci"
      version = "2.11.1"
  }
}
```

versions.tf

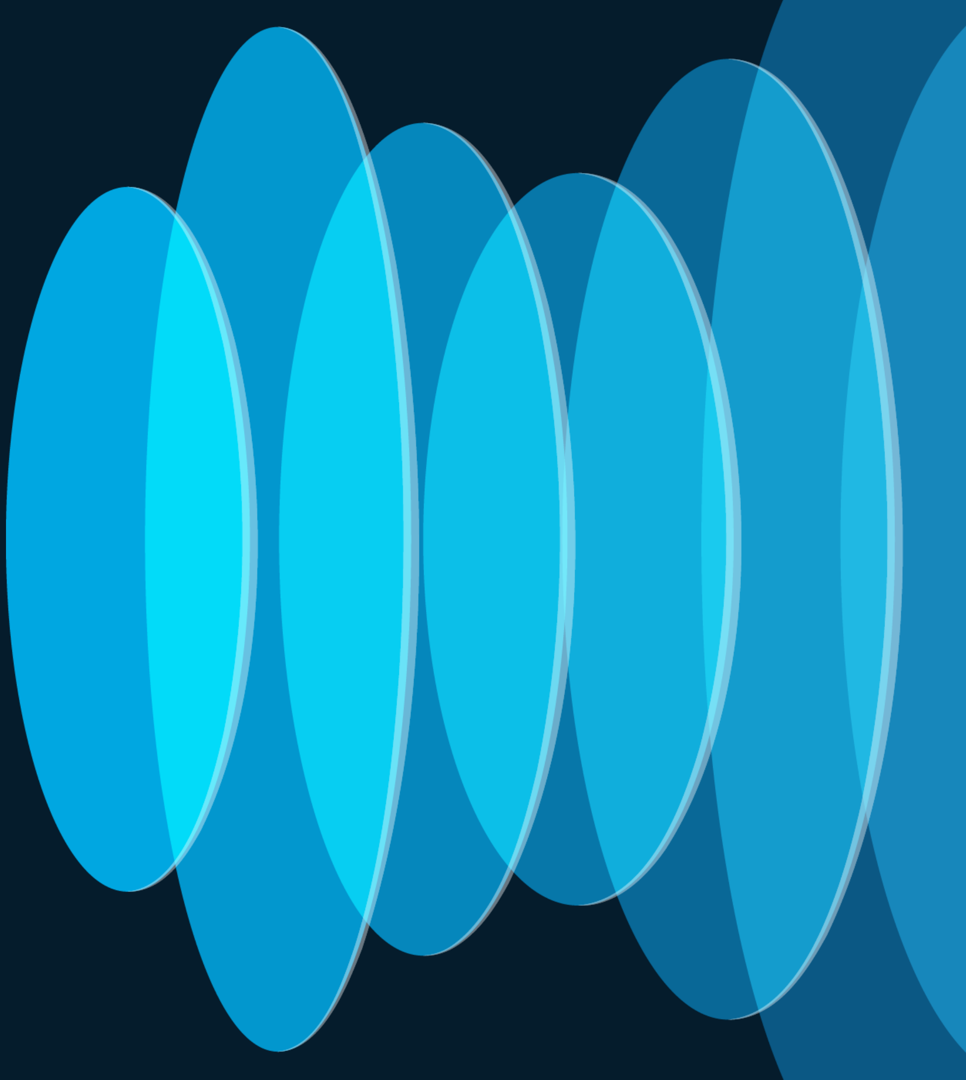
```
variable "tenant_name" {
  type = string
}

variable "ap_name" {
  type = string
}

variable "vrf_name" {
  type = string
}

variable "epg_name" {
  type = string
}

variable "bd_name" {
  type = string
}
...
```

variables.tf
(match input variables)

```
resource "aci_tenant" "mod_tenant" {
  name        = var.tenant_name
  description = "Created with Terraform Modules"
}

resource "aci_vrf" "mod_vrf" {
  tenant_dn   = aci_tenant.mod_tenant.id
  name        = var.vrf_name
  description = "Created with Terraform Modules"
}

resource "aci_application_profile" "mod_ap" {
  tenant_dn = aci_tenant.mod_tenant.id
  name      = var.ap_name
  description = "Created with Terraform Modules"
}
...
```

main.tf

Deploying Infrastructure as Code with Terraform



Terraform – CLI commands

`terraform init`

- Download and Installs plugins for configured providers
- Must initialize before plan/apply
- Creates a provider "lock" file

```
terraform {
  required_providers {
    aci = {
      source = "CiscoDevNet/aci"
    }
  }
}
```

`terraform plan`

- Scans the current directory for the configuration (.tf & .tfvars extension)
- Determines what actions are necessary to achieve the desired state
- Preview your changes – no changes made (Dry Run)

`terraform apply`

- Scans the current directory for the configuration (.tf & .tfvars extension)
- Preview your changes (can bypass with `-auto-approve`)
- Applies the configuration to targets (upon approval "yes")

`terraform destroy`

- Scans the state file for what to "destroy"
- Preview your deletions
- Infrastructure is destroyed
- Can be specific with "-target"

Terraform – CLI commands

`terraform fmt`

- Formats Terraform configuration files in directory

`terraform show`

- Show the state file in a readable format
- Can also read a specific state file (path)

`terraform state`

- Advanced State Management
- `show <resource>` - Shows a particular resource
- `list` – Lists all resources in current state file
- `rm <instance>` – Remove an instance from the state file
- `mv` – Move an item. Good for renaming resources

`terraform validate`

- Verifies correctness of Terraform configuration files (*.tf)
- Checks syntax
- Can be used to solve configuration of errors

Terraform Demo

Terraform Import

Terraform Import

- Terraform – Great IaC tool
- But what about objects that were not provisioned with Terraform?
- Terraform `import` CLI command
  - Import Infrastructure for Terraform to manage
  - Imports to Terraform state file
  - Does not generate the associated configuration

```
terraform import <resource_type.resource_name> <full path to Dn>
```

- Issues with this method
  - Only imports single object - Must be run for all child objects
  - Can get very tedious

Terraform Import Blocks

- Declarative method to Import Infrastructure to Terraform
  - Create import block configuration for all objects to import
  - Create the associated Terraform configurations for that object
    - `terraform plan -generate-config-out=new_tenant.tf`

```
# Import VRF under Tenant from APIC
import {
  id = "uni/tn-tf_test_import/ctx-tf_test_import_vrf"
  to = aci_vrf.import_vrf_example
}

# Import Tenant from APIC
import {
  id = "uni/tn-tf_test_import"
  to = aci_tenant.import_tenant_example
}
...
```

(import.tf)

Terraform Import

- `terraform plan`

```
> terraform plan

...
Plan: 2 to import, 0 to add, 0 to change, 0 to destroy.
```

- `terraform apply`

```
> terraform apply -auto-approve

...
Apply complete! Resources: 2 imported, 0 added, 0 changed, 0 destroyed.
```

- `terraform state list`

```
> terraform state list
aci_tenant.import_tenant_example
aci_vrf.tf_test_import_vrf
```

Key Takeaways

Infrastructure as Code with

- Install and test Terraform
  - Available for most platforms
- Think big…..start small
  - Automate the simple, then build into more complex tasks
  - Try different things.
- Writing IaC with Terraform is easy
  - No special programming skills needed
- Terraform Resources for most common tasks
- Robust APIC/MSO REST API makes automation easy and scalable

More information – Other sessions/labs

- LABDCN-1776 (Walk in Lab - Intro to Terraform with ACI)
- BRKDCN-2673 - Nexus-as-Code - Kickstart your automation with ACI
- DEVWKS-2931 - Making your ACI Automation as modular as LEGO bricks using Terraform Modules
- IBODCN-1003 - An Interactive Conversation on ACI Automation through Ansible and Terraform

More information – Ansible/Terraform

- https://www.terraform.io/
- https://registry.terraform.io/providers/CiscoDevNet/aci/latest/docs
- https://registry.terraform.io/providers/CiscoDevNet/mso/latest
- https://developer.cisco.com/automation-terraform/
- https://developer.hashicorp.com/terraform/language/modules
- https://github.com/rafmuller/brkdcn-ans-terra

Complete Your Session Evaluations

Complete a minimum of 4 session surveys and the Overall Event Survey to be entered in a drawing to **win 1 of 5 full conference passes** to Cisco Live 2025.

Earn 100 points per survey completed and compete on the Cisco Live Challenge leaderboard.

Level up and earn **exclusive prizes!**

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

Continue your education

- Visit the Cisco Showcase for related demos
- Book your one-on-one Meet the Engineer 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

The bridge to possible

Thank you

#CiscoLive