Composable Infrastructure with ACI and Terraform

How to share network knowledge in a DRY fashion

Nicolas Vermande, Technical Marketing Engineer – IBNG
@nvermande

cisco Live!

cisco

About Me

5 years at Cisco

Big focus on Cloud Native

All things OSS and ACI

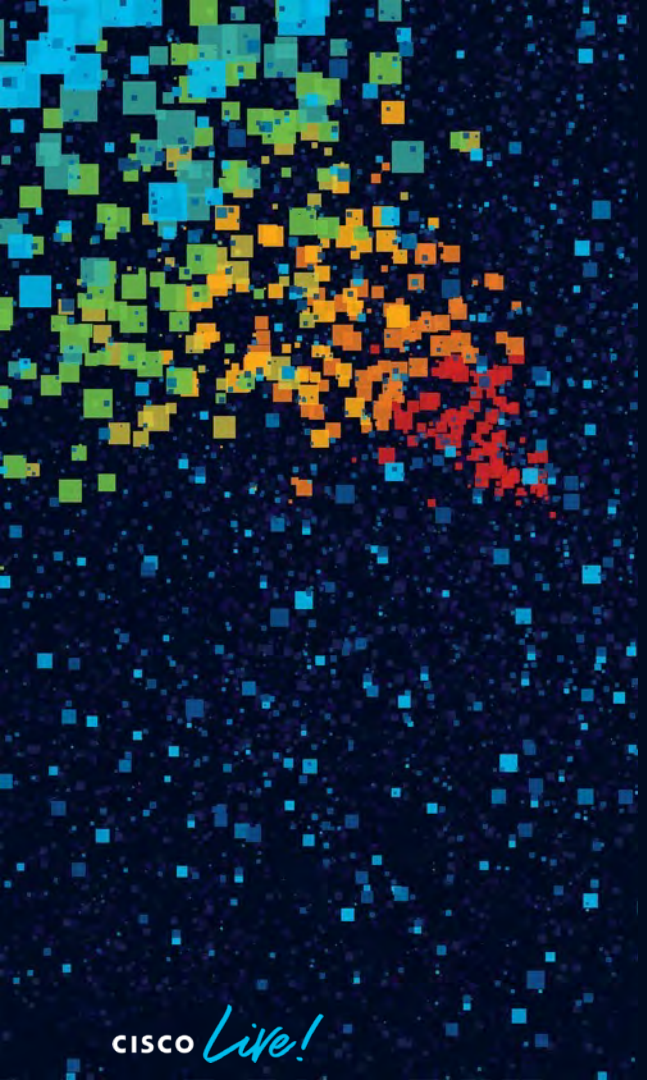
Automation junkie

Love coding

Double VMware VCDX

CCIE

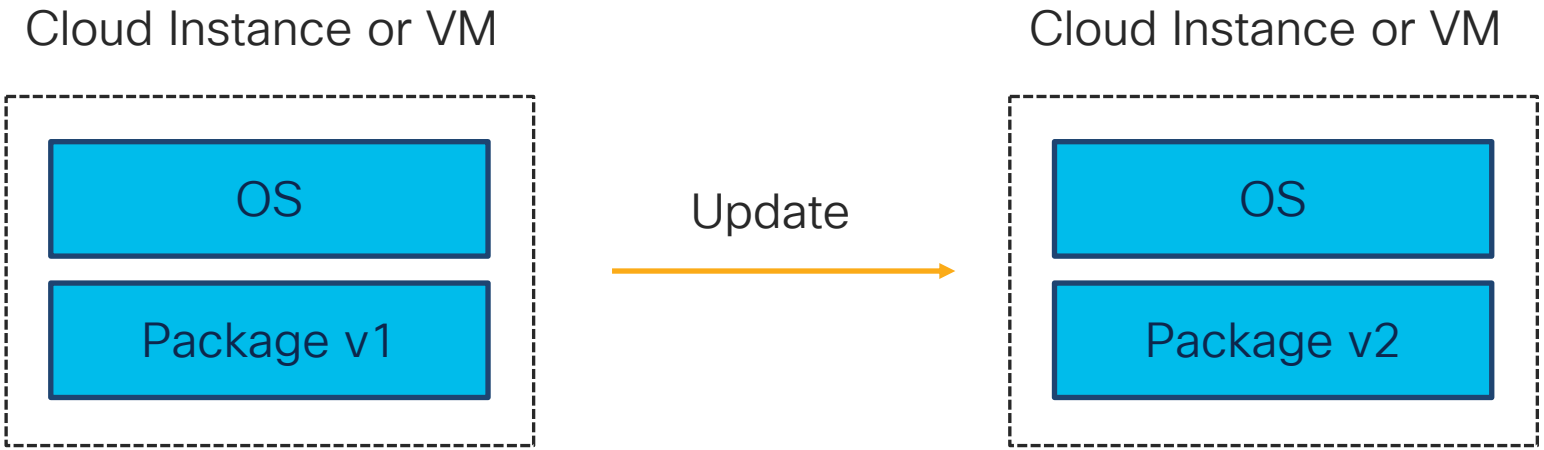
Groove Metal addict



Agenda

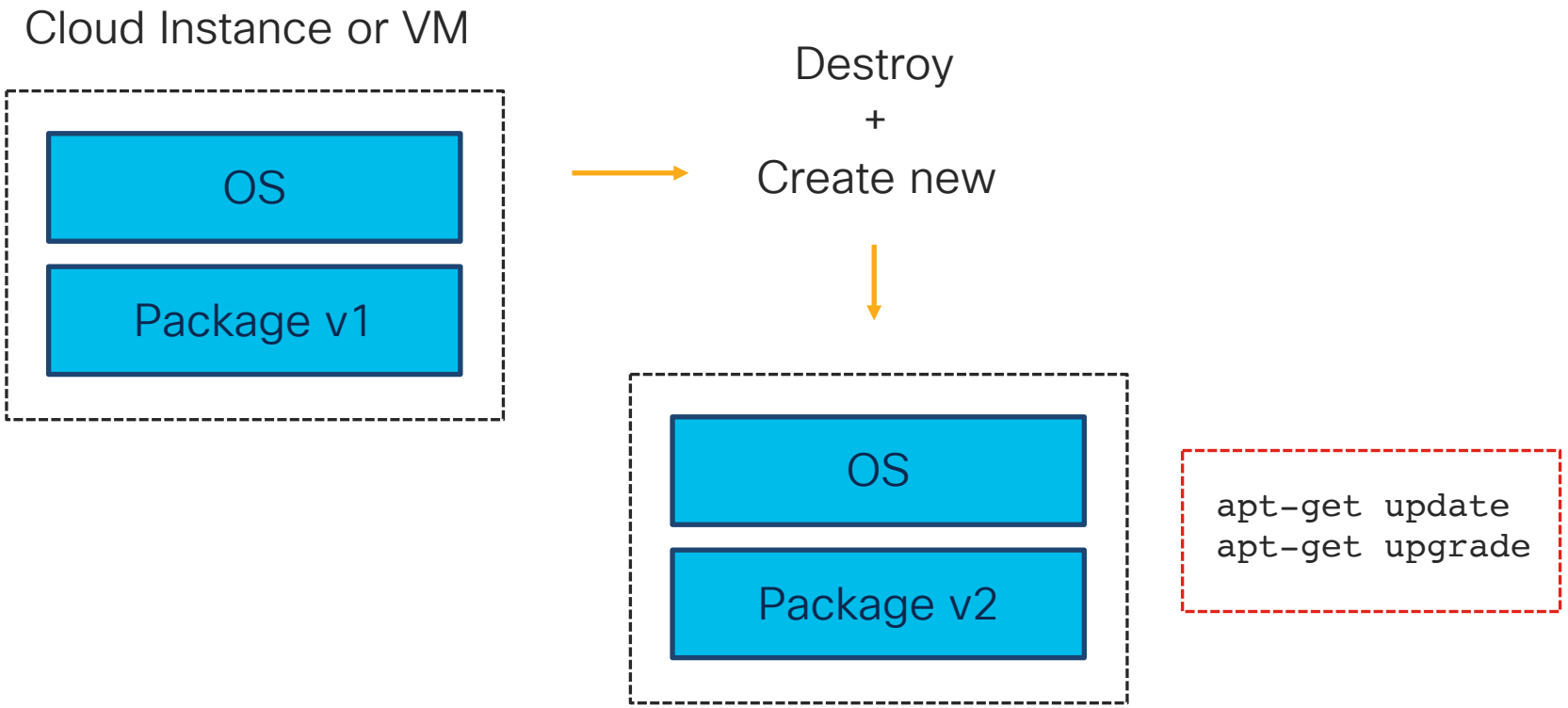
- Introduction to Terraform
- Optimize ACI Operations
- Practical code example
- Call to Actions

Immutable Infrastructure



Immutable Infrastructure

Immutable Infrastructure



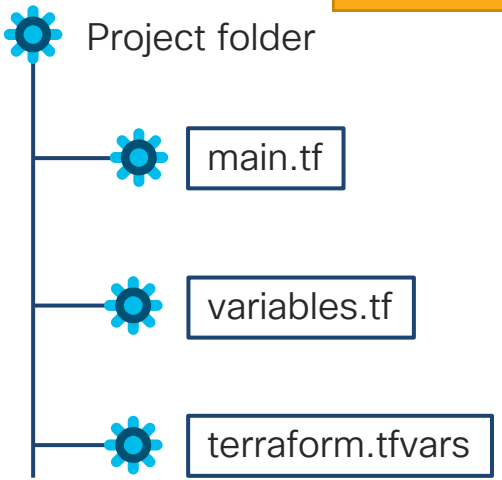
Immutable Infrastructure

- No Customization
- Stateless System
- Data needs to be externalized

Composable Infrastructure

- Re-usable modules
- Import of existing components
- Dependency management
- Sharing of knowledge
- Versions linked to environment

Terraform Terminology



Resource “class”

Resource “instance”

```
51  resource "aci_tenant" "terraform_ten" {
52    name = var.tenant_name
53  }
```

`name` is an attribute of the instance

```
1  variable "tenant_name" {}
```

```
1  tenant_name = "tenant-01"
```

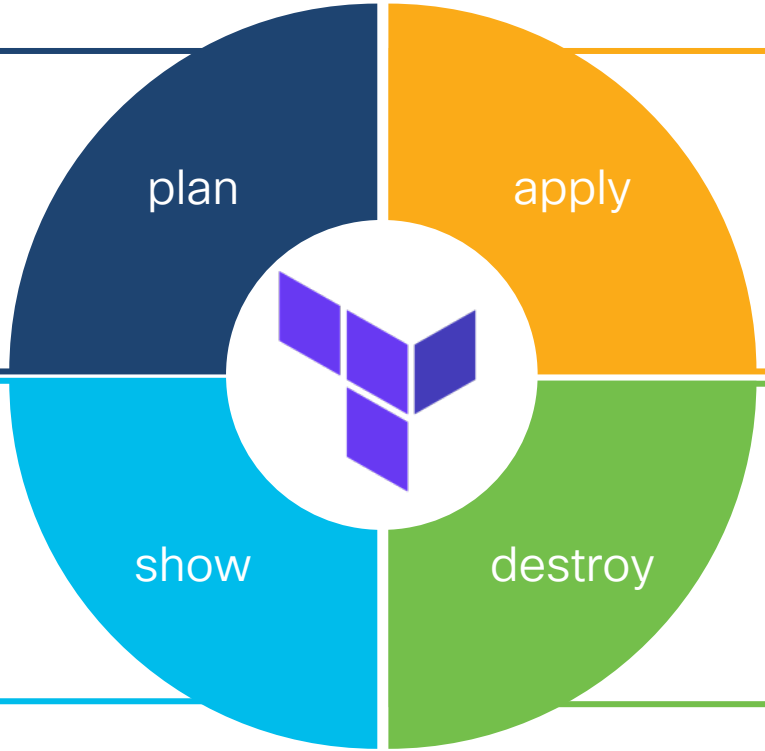
Main Commands

Execution Plan

Display actions to be performed to reach the desired state

Execute Actions

Modify state of resources in accordance to the plan



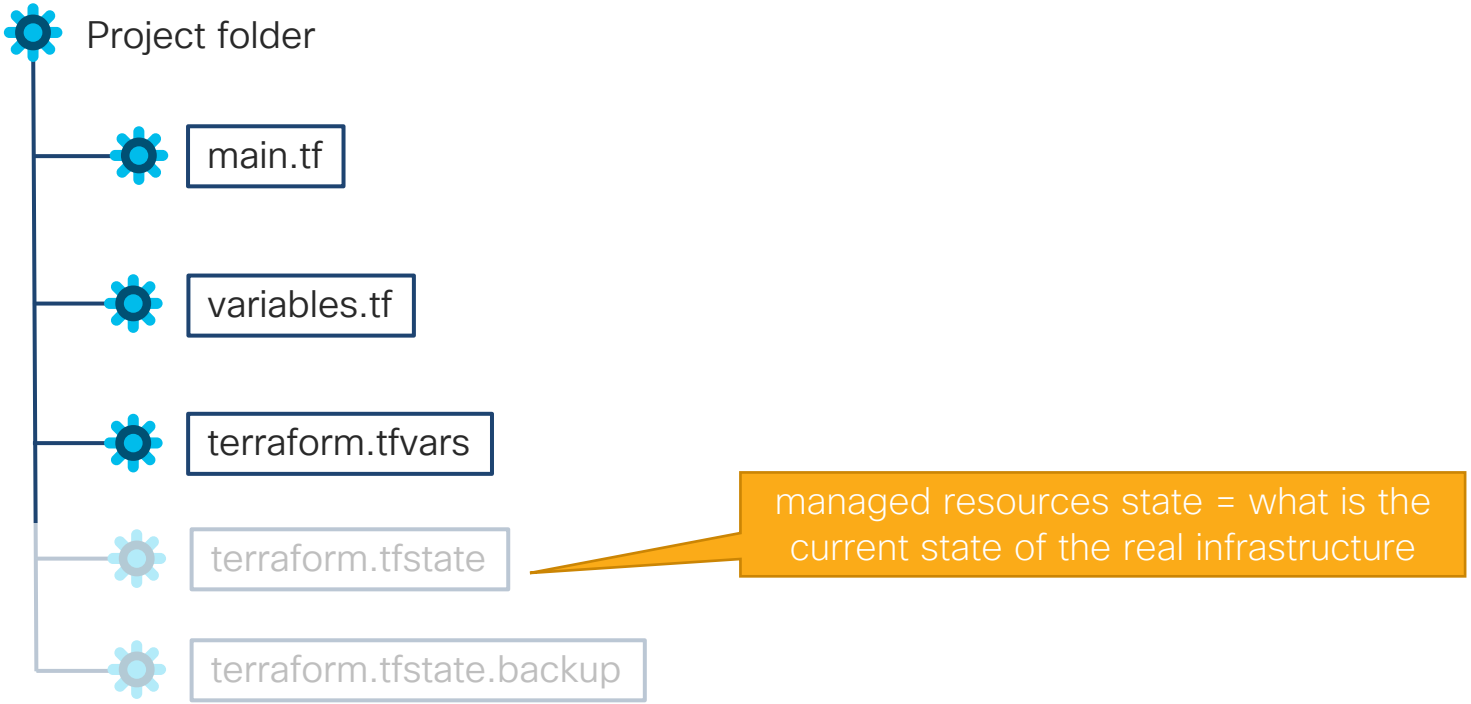
Current State

Current output from the state file

Remove Resources

Delete infrastructure resources present in the state file

Where is the state file?



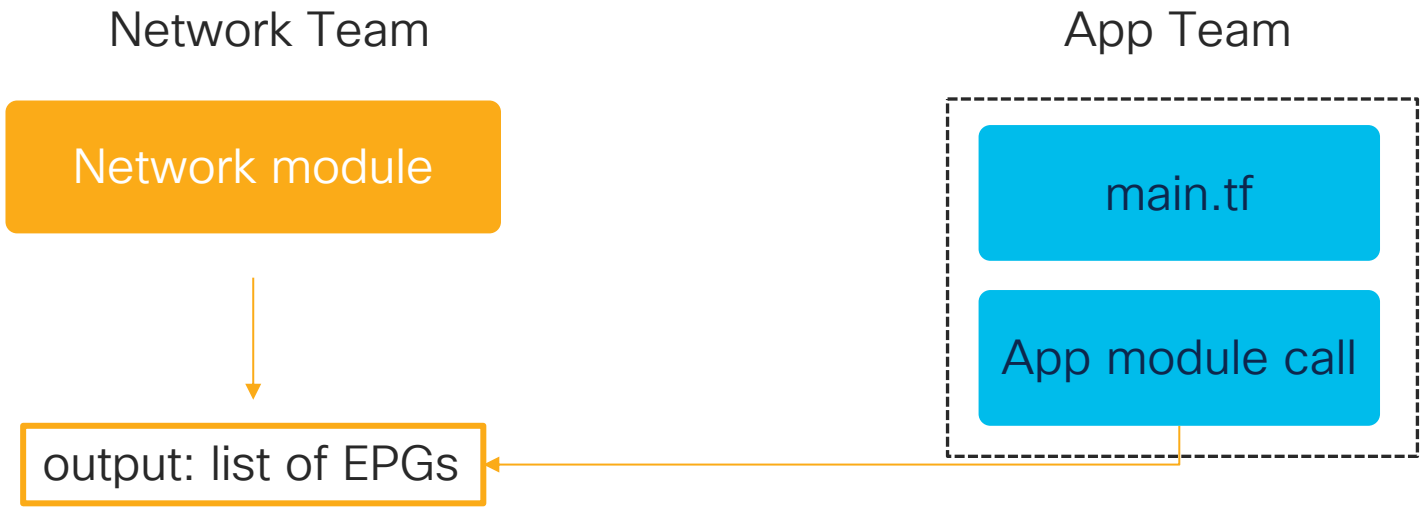
ACI Operations with Terraform

- Use Remote Backend (Terraform Cloud has it by default)
- Use Git or any other VCS to managed configuration files
- Few resources per TF file is better
- Use Data Sources to avoid unexpected ACI construct deletion
- Create outputs for other teams to reference global ACI network constructs (e.g.: Tenant, VRF, etc)

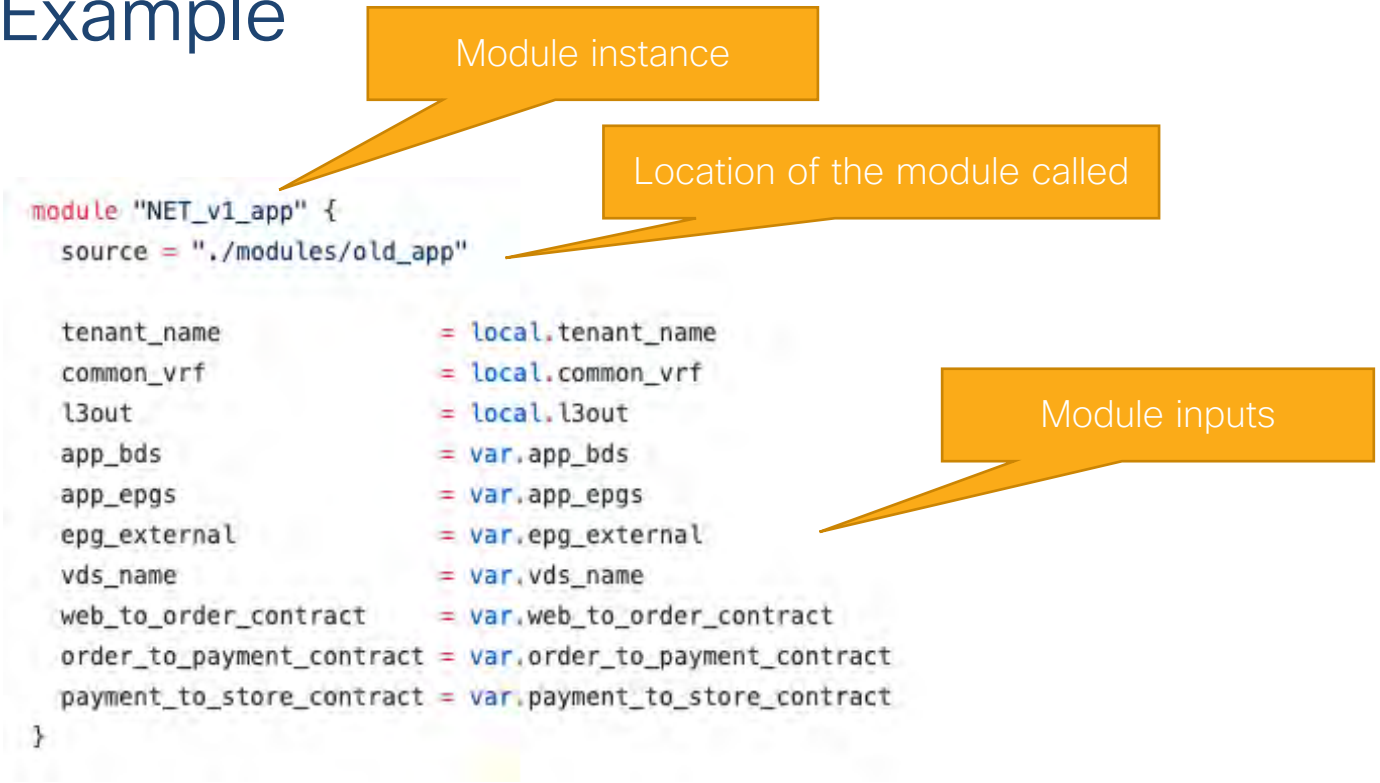
Keep it DRY

- Modules are like functions and provide reusable components
- Modules are Terraform configuration files within a folder (nothing more), but variables are not usable in main TF file outside of the module stanza
- Modules only take inputs, return outputs and contain resources.

Practical Example



Practical Example



Practical Example

```
module "app_vm" {
  source = "./modules/create_vm"

  vsphere_user            = var.vsphere_user
  vsphere_password        = var.vsphere_password
  vsphere_server          = var.vsphere_server
  vsphere_datacenter      = var.vsphere_datacenter
  vsphere_datastore       = var.vsphere_datastore
  vsphere_compute_cluster = var.vsphere_compute_cluster
  net_mgmt                = local.vm_network
  vsphere_template        = var.vsphere_template
  vm_name                 = var.vm_prefix
  vcpu                    = var.vcpu
  memory                  = var.memory
  folder                  = var.folder
  domain_name             = var.domain_name
  vm_ip_address_start     = var.vm_ip_address_start
  vm_cidr                 = var.vm_cidr
  gateway                 = var.gateway
  dns_list                = var.dns_list
  dns_search              = var.dns_search
  vm_depends_on           = module.NET_v1_app.network
  multi_nic               = "false"
  is_linked_clone         = "false"

}
```

reference to previous module output

Practical Example

- Output defined in the network module:

```
output "network" {
    value = aci_application_epg.app_epgs
}
```

Call to Actions

- Start using Terraform to create basic ACI objects
- Create you first module that can be re-used for repetitive tasks
- Explore Terraform Cloud and workspaces
- Think about ACI services you can provide to other teams as modules
- Think about ACI outputs that other teams will need
- Code can be found here: https://github.com/vfiftyfive/tf-aci-app-centric-mig

Thank you

Possibilities

#CiscoLive | #DevNetDay