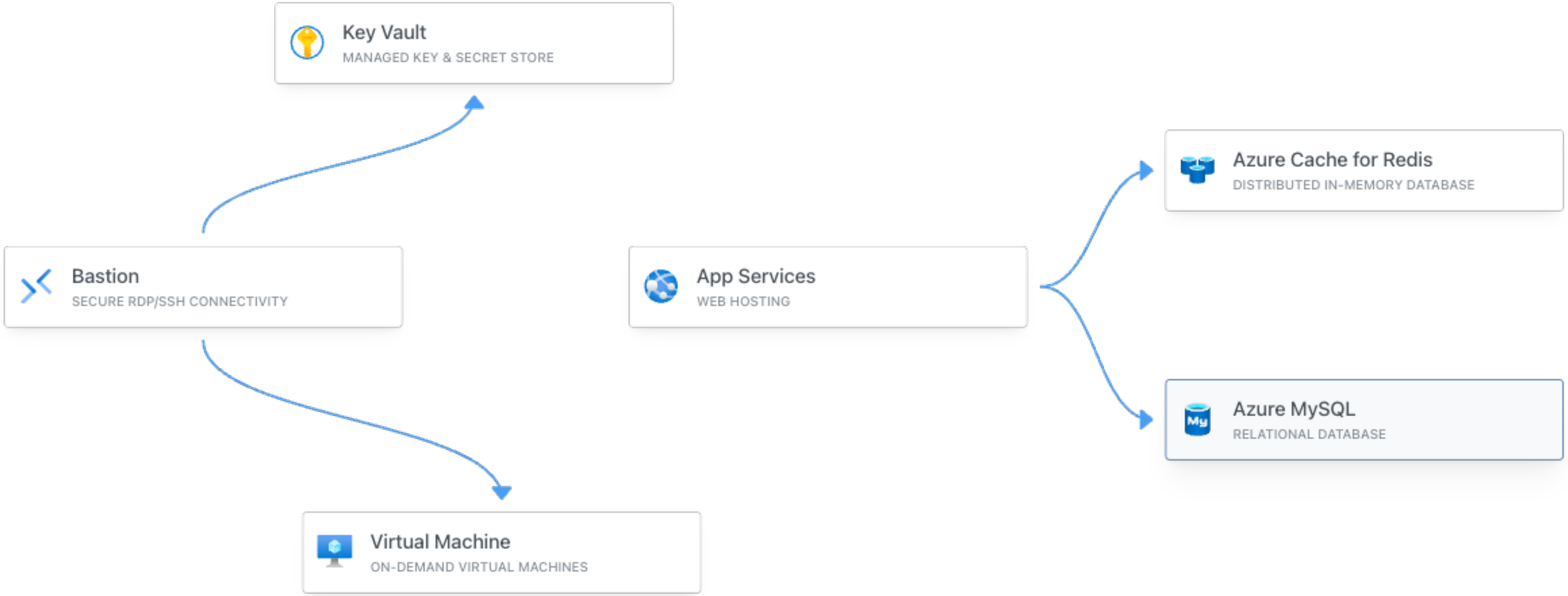
Scaling Terraform Configurations

To scale the infrastructure, scalable code one needs.

— Yoda, DevOps Master

Azure Architecture



Phase 1: Monolith

```
terraform/
┣ terraform.tf
┣ terraform.tfvars
┗ terraform.tfstate
```

```
### Dev ###
resource "azurerm_resource_group" "dev" {
  name     = "rg-${var.project_name}-dev"
  location = var.location
}

resource "azurerm_virtual_network" "dev" {
  name                = "vnet-${var.project_name}-dev"
  address_space       = ["10.0.0.0/16"]
  resource_group_name = azurerm_resource_group.dev.name
  location            = azurerm_resource_group.dev.location
}

resource "azurerm_subnet" "core_dev" {
  name                 = "core"
  resource_group_name  = azurerm_resource_group.dev.name
  virtual_network_name = azurerm_virtual_network.dev.name
  address_prefixes     = ["10.0.1.0/24"]
}
```

```
### Dev ###
resource "azurerm_resource_group" "dev" {
  name = "rg-${var.project_name}-dev"
  location = var.location
}

# ...

### Prod ###
resource "azurerm_resource_group" "prod" {
  name = "rg-${var.project_name}-prod"
  location = var.location
}

# ...
```

Characteristics

» Single state

» Hard coded configuration

» All definitions in a single file

» Duplication

Quick and dirty approach

Phase 2: Multi-Monolith

```
terraform/
  environments/
    dev/
      terraform.tf
      terraform.tfvars
    prod/
      terraform.tf
      terraform.tfvars
```

Characteristics

» Environment isolation

» Multiple configuration files

» Duplication among environments

» 1:1 relationship between environments and state files

Better, but still not scalable

Phase 3: Modules

A component is a logical grouping of resources that work together to provide a higher-level service.

Each component has a corresponding module. *Modules* are used to encapsulate the configuration of a component and are reusable across environments.

```
terraform/
  environments/
    dev/
      terraform.tf
      terraform.tfvars
    prod/
      terraform.tf
      terraform.tfvars
  modules/
    compute/
      main.tf
    core/
      main.tf
    database/
      main.tf
```

```
module "core" {
  source = "../../modules/core"
  vnet_name = "vnet-${var.project_name}-dev"
  vnet_cidr = "10.0.0.0/16"
  ...
}

module "compute" {
  source = "../../modules/compute"
  ...
}

module "database" {
  source = "../../modules/database"
  ...
}
```

For each module split the configuration into separate files:

- `main.tf` contains the main configuration of the module
- `variables.tf` contains the input variables of the module
- `outputs.tf` contains the output variables of the module

Inputs and outputs define the interface of the module.

```
terraform/
  modules/
    compute/
      main.tf
      variables.tf
      outputs.tf
    core/
      main.tf
      variables.tf
      outputs.tf
    database/
      main.tf
      variables.tf
      outputs.tf
```

For each environment split the configuration into separate files:

- `main.tf` contains the main configuration of the environment
- `variables.tf` contains the input variables of the environment
- `outputs.tf` contains the output variables of the environment
- `terraform.tfvars` contains the values of the input variables
- `terraform.tf` contains configuration about terraform version, providers, and state

```
terraform/
  environments/
    dev/
      main.tf
      variables.tf
      outputs.tf
      terraform.tf
      terraform.tfvars
    prod/
      main.tf
      variables.tf
      outputs.tf
      terraform.tf
      terraform.tfvars
```

Characteristics

» Directory restrucutre

» Multiple configuration files

» DRY principle

» Reusable modules

First step to reusability and maintainability

Phase 4: Multilayer Modules

A module can be used to encapsulate a single resource, a group of resources, a higher-level component, or an infrastructure stack.

Split modules into two categories, base and composite:

» **Base modules** are reusable modules that encapsulate the configuration of low-level infrastructure.

» **Composite modules** are modules that use other modules to create a higher-level component.

```
modules/
  common/  # Base modules
    network/
      main.tf
      variables.tf
      outputs.tf
    virtual-machine/
      main.tf
      variables.tf
      outputs.tf
  project-x/  # Composite modules, project specific
    compute/
      main.tf
      variables.tf
      outputs.tf
    core/
      main.tf
      variables.tf
      outputs.tf
    database/
      main.tf
      variables.tf
      outputs.tf
```

A base module can be used in multiple composite modules, and a composite module can be used in multiple environments.

```
# File: modules/project-x/database/main.tf

module "redis" {
  source = "../../common/redis"
  ...
}

module "mysql" {
  source = "../../common/mysql"
  ...
}
```

```
# File: environment/dev/main.tf

module "core" {
  source = "../../modules/project-x/core"
  ...
}

module "compute" {
  source = "../../modules/project-x/compute"
  ...
}

module "database" {
  source = "../../modules/project-x/database"
  ...
}
```

A base module can contain submodules of its own. These are used by the base module, but can also be referenced on their own.

```
network/ # Base module: creates at least one virtual network with one subnet
  main.tf
  variables.tf
  outputs.tf
  modules/ # Submodules: used by the base module, but can also be used on their own
    subnet/
      main.tf
      variables.tf
      outputs.tf
    route-table/
      main.tf
      variables.tf
      outputs.tf
    network-security-group/
      main.tf
      variables.tf
      outputs.tf
```

```
# File: modules/common/network/main.tf

resource "azurerm_virtual_network" "vnet" {
  ...
}

module "subnets" {
  for_each = var.subnets
  source   = "./modules/subnets"
  ...
}

module "nsgs" {
  for_each = var.nsgs
  source   = "./modules/nsgs"
  ...
}
```

Organize base modules to repositories. This allows for better reuse and sharing of infrastructure code.

Approach 1: Monorepo

```
terraform-base-modules/
  network/
    main.tf
    variables.tf
    outputs.tf
  virtual-machine/
    main.tf
    variables.tf
    outputs.tf
  redis/
    main.tf
    variables.tf
    outputs.tf
  ...
```

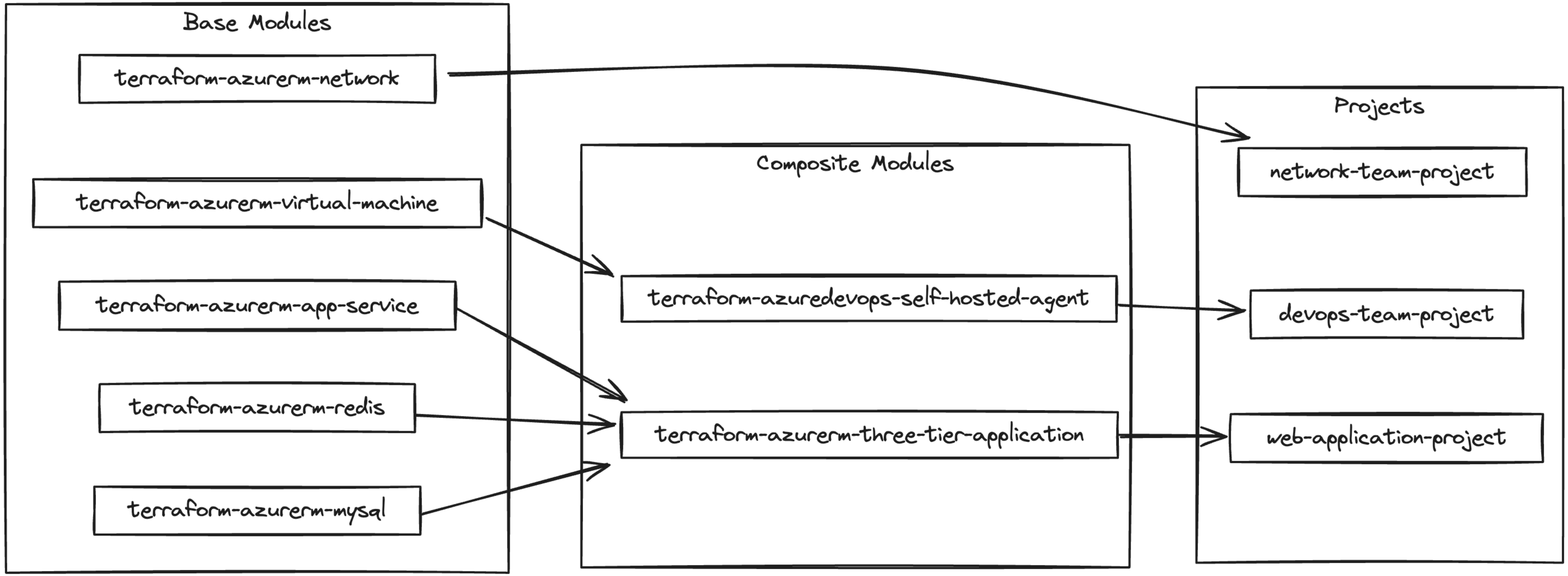
Approach 2: Multirepo (one repository per module)

```
terraform-azurerm-network/
  main.tf
  variables.tf
  outputs.tf

terraform-azurerm-virtual-machine/
  main.tf
  variables.tf
  outputs.tf

terraform-azurerm-redis/
  main.tf
  variables.tf
  outputs.tf
```

Organize composite modules to repositories.


Base Modules
terraform-azurerm-network
terraform-azurerm-virtual-machine
terraform-azurerm-app-service
terraform-azurerm-redis
terraform-azurerm-mysql
Composite Modules
terraform-azuredevops-self-hosted-agent
terraform-azurerm-three-tier-application
Projects
network-team-project
devops-team-project
web-application-project

Characteristics

» Nested modules

» Even DRYier

» Maintenance of multiple repositories

» Reusability

» Versioning

Starting point for scaling Terraform configurations

Phase 5: Stacks

If the configuration among environments is similar, and the only difference is the values of the input variables, then use a single module to manage all environments.

This module can be thought of as an infrastructure stack, and it can be used to manage multiple environments.

```
# File: environment/dev/main.tf

module "stack" {
  source = "../../stacks/project-x"
  ...
}
```

```
terraform/
  environments/
    dev/ ...
    prod/ ...
  stacks/
    project-x/ ...
  modules/
    common/ ...
    project-x/ ...
```

Characteristics

» Single stack for multiple environments

» Updates to the stack affect all environments

Great approach for medium-sized projects

Phase 6: Services

Each high-level component gets its own state file. This allows for better isolation and control over the infrastructure.

Share information between different high-level components using remote state data sources.

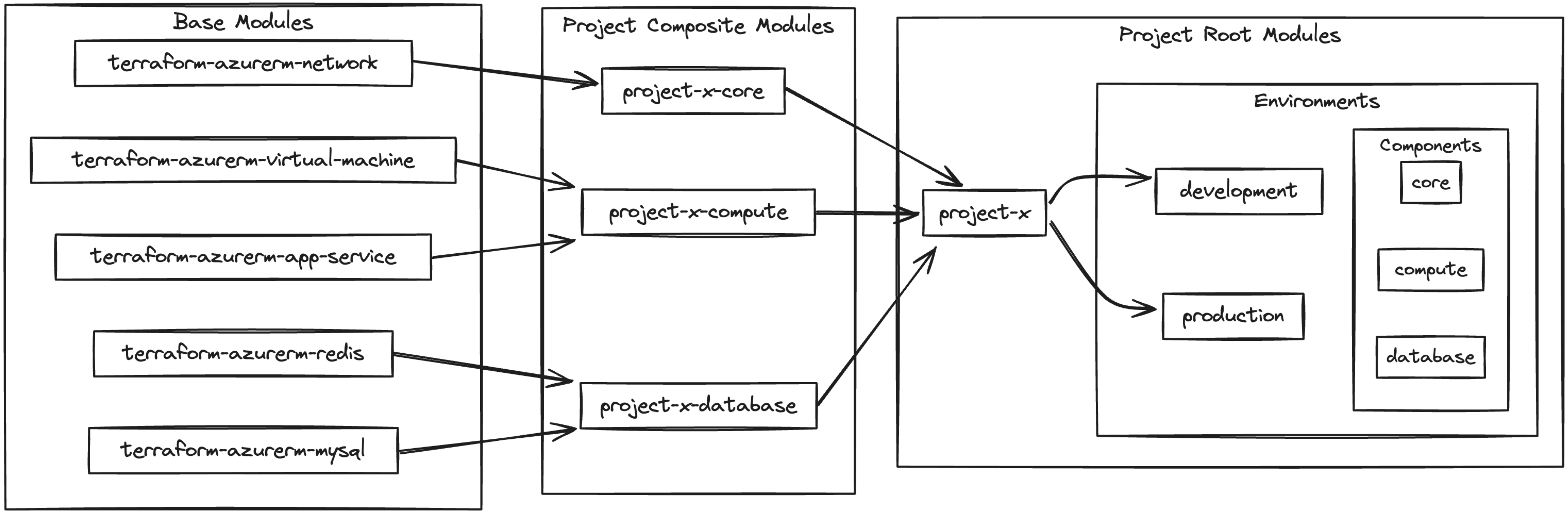
Organize composite modules to repositories. Helpful when different teams are responsible for different parts of the infrastructure.

```
project-x-core/
  main.tf
  variables.tf
  outputs.tf

project-x-compute/
  main.tf
  variables.tf
  outputs.tf

project-x-database/
  main.tf
  variables.tf
  outputs.tf

project-x/
  environments/
    dev/
      core/
        main.tf
        variables.tf
        outputs.tf
      ...
    prod/
      ...
```


Base Modules
terraform-azurerm-network
terraform-azurerm-virtual-machine
terraform-azurerm-app-service
terraform-azurerm-redis
terraform-azurerm-mysql
Project Composite Modules
project-x-core
project-x-compute
project-x-database
Project Root Modules
project-x
Environments
development
production
Components
core
compute
database

Characteristics

» Independent management for each high-level component

» A lot more complexity and effort

» Order of execution matters

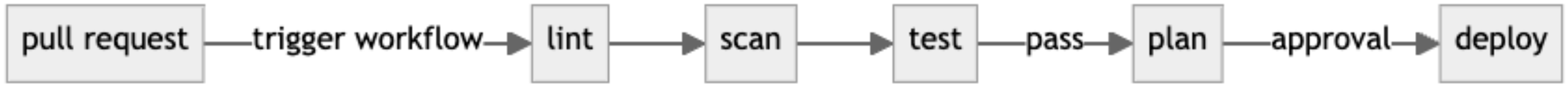
» Separation of responsibility

» Scalability

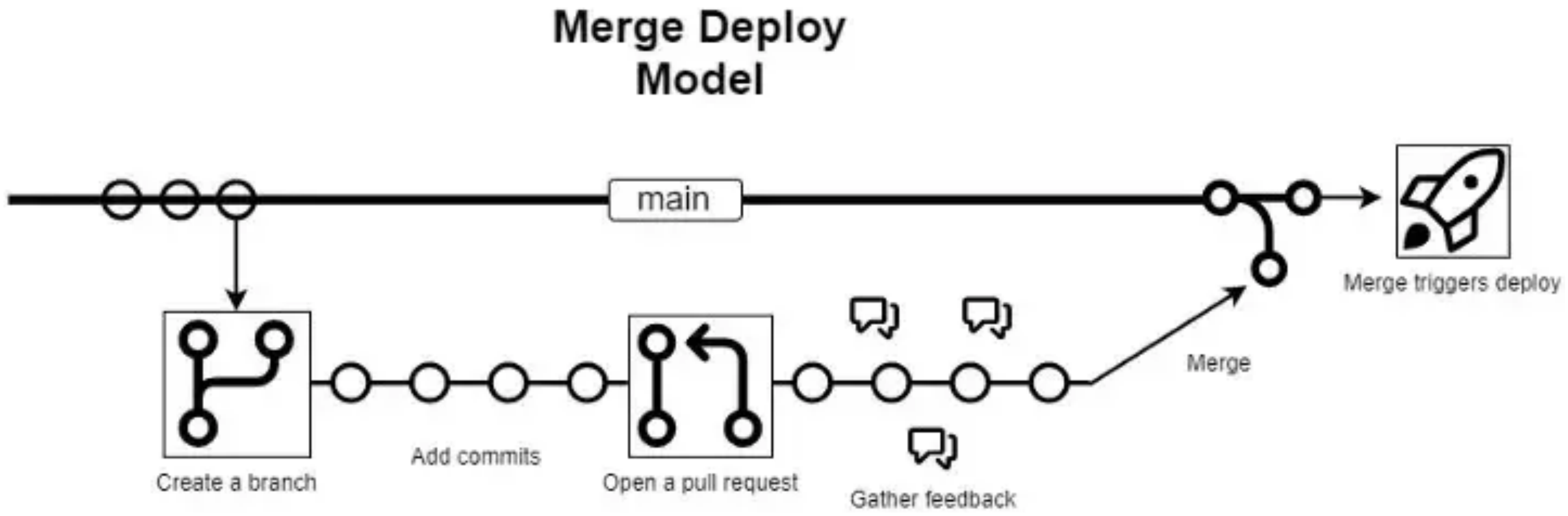
Complex but efficient approach for large projects

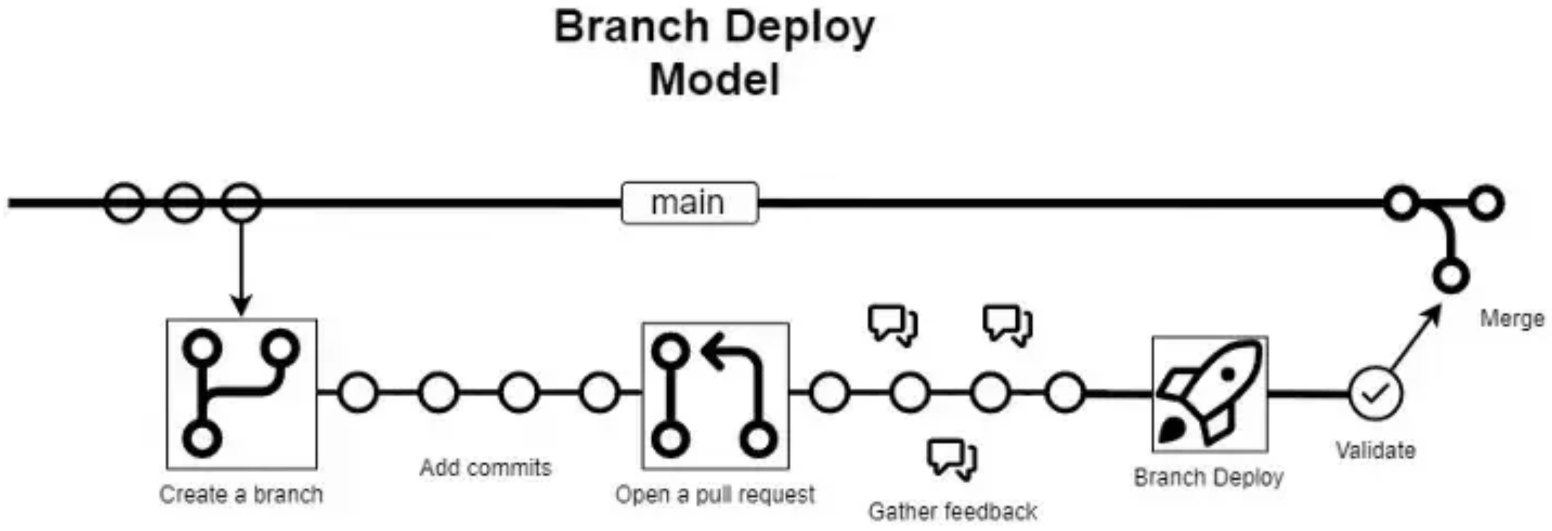
Phase *: CI/CD

Create pipelines to make infrastructure changes



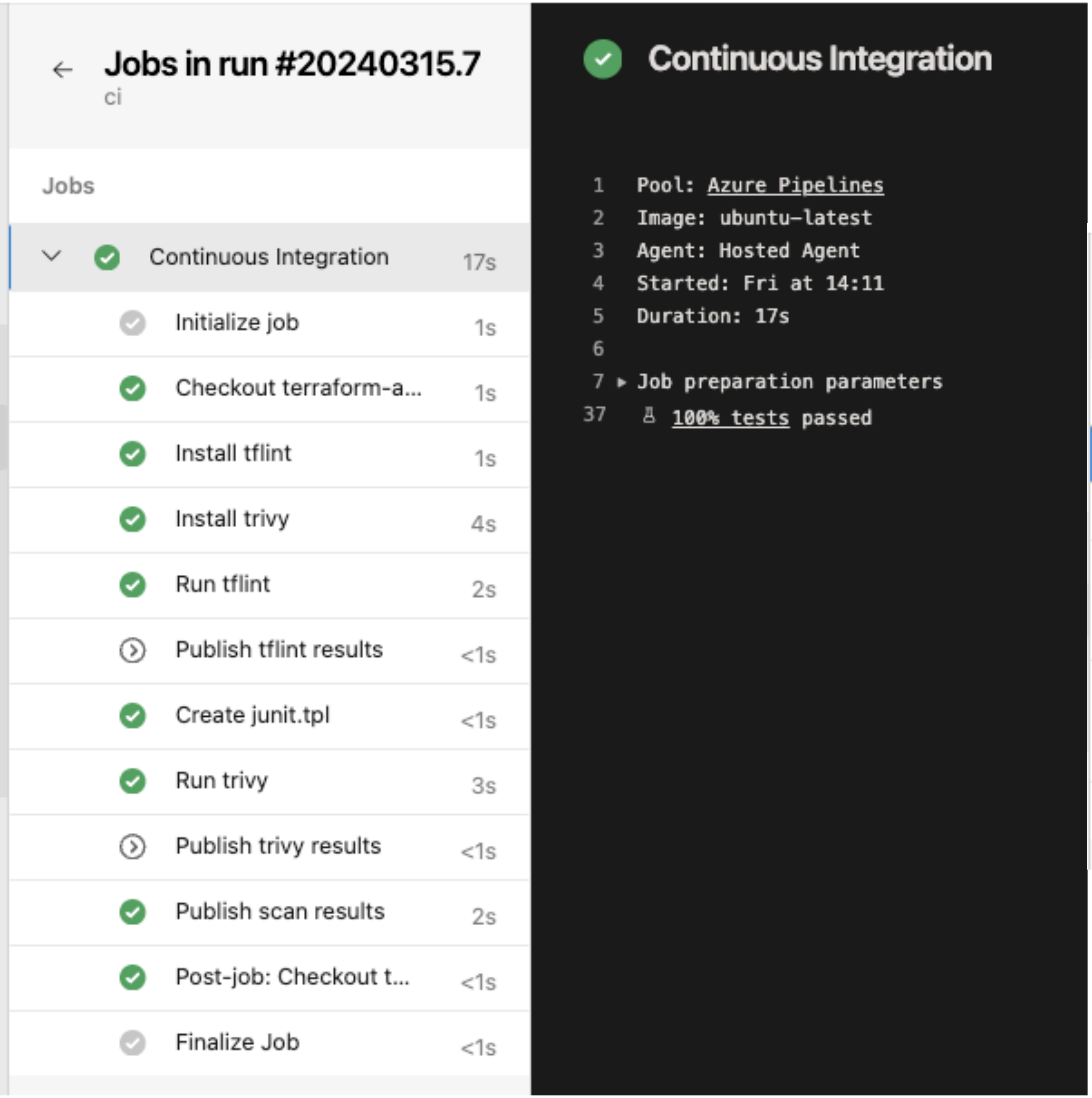
Choose a deployment model



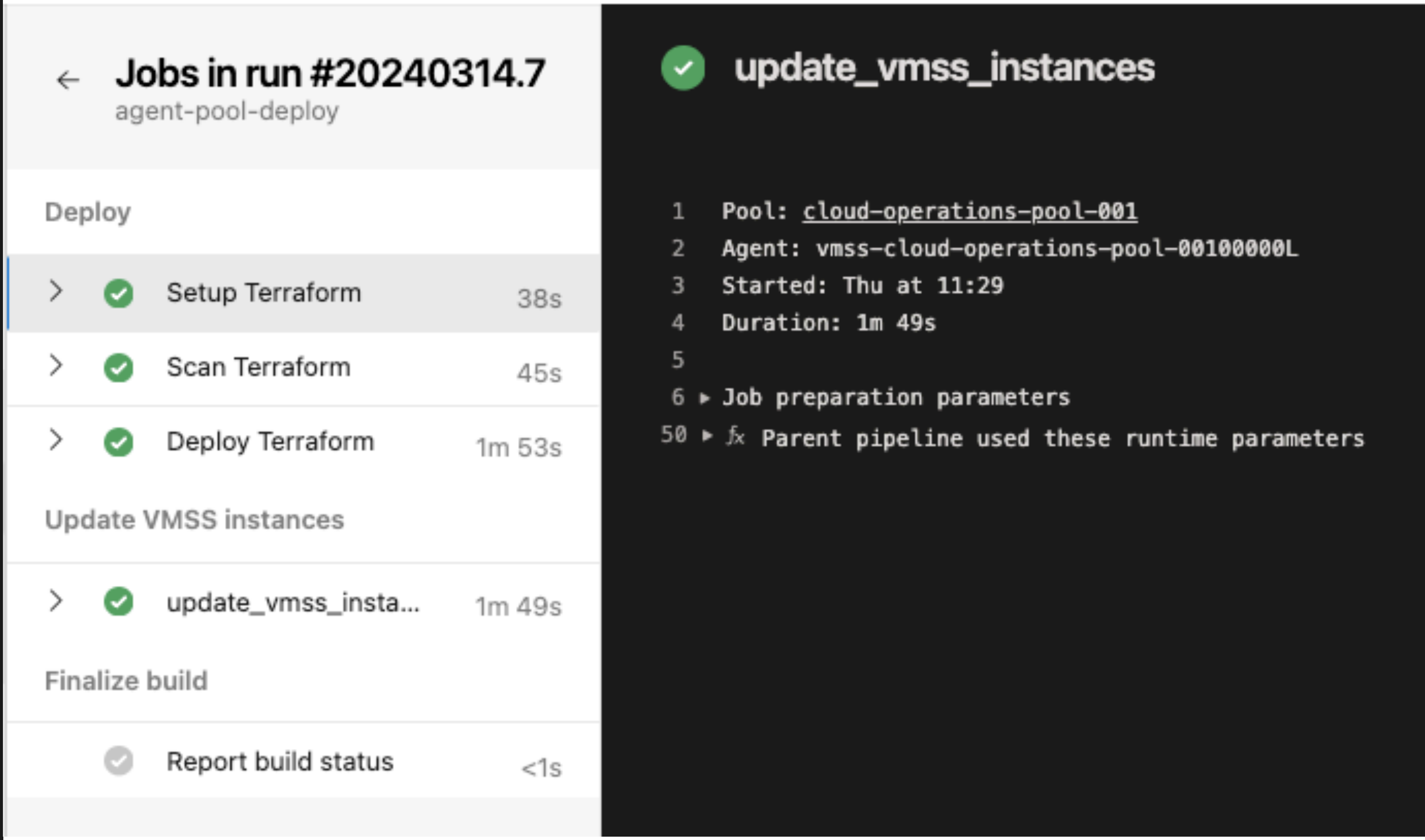


https://github.blog/2023-02-02-enabling-branch-deployments-through-issueops-with-github-actions/

CI on module development; why not?



Jobs in run #20240315.7
ci
Jobs
Continuous Integration 17s
Initialize job 1s
Checkout terraform-a... 1s
Install tflint 1s
Install trivy 4s
Run tflint 2s
Publish tflint results <1s
Create junit.tpl <1s
Run trivy 3s
Publish trivy results <1s
Publish scan results 2s
Post-job: Checkout t... <1s
Finalize Job <1s
Continuous Integration
1 Pool: Azure Pipelines
2 Image: ubuntu-latest
3 Agent: Hosted Agent
4 Started: Fri at 14:11
5 Duration: 17s
6
7 ▸ Job preparation parameters
37 100% tests passed



Jobs in run #20240314.7
agent-pool-deploy
Deploy
Setup Terraform 38s
Scan Terraform 45s
Deploy Terraform 1m 53s
Update VMSS instances
update_vmss_insta... 1m 49s
Finalize build
Report build status <1s
update_vmss_instances
1 Pool: cloud-operations-pool-001
2 Agent: vmss-cloud-operations-pool-00100000L
3 Started: Thu at 11:29
4 Duration: 1m 49s
5
6 ▸ Job preparation parameters
50 ▸ fx Parent pipeline used these runtime parameters

Run 180 - Terraform Scan Results

Run summary Test results Filter

Create bug | Update analysis

Outcome	Test Case Title	P.	D.	O	C.	M	Error message
Passed	[NONE][CKV_AZURE_118] Ensure that Network Interfaces disable I...	0.	0.		N.		
Passed	[NONE][CKV_AZURE_179] Ensure VM agent is installed	0.	0.		N.		
Passed	[NONE][CKV_AZURE_92] Ensure that Virtual Machines use manage...	0.	0.		N.		
Passed	[NONE][CKV_AZURE_178] Ensure linux VM enables SSH with keys f...	0.	0.		N.		
Passed	[NONE][CKV_AZURE_50] Ensure Virtual Machine Extensions are no...	0.	0.		N.		
Passed	[NONE][CKV_AZURE_119] Ensure that Network Interfaces don't use...	0.	0.		N.		
Passed	[NONE][CKV2_AZURE_39] Ensure Azure VM is not configured with ...	0.	0.		N.		
Failed	[NONE][CKV_AZURE_1] Ensure Azure Instance does not use basic a...	0.	0.		N.		Ensure Azure Instance does not use basic authentication(Use SSH Key Instead)
Failed	[NONE][CKV_AZURE_149] Ensure that Virtual machine does not en...	0.	0.		N.		Ensure that Virtual machine does not enable password authentication
Failed	terraform_unused_declarations	0.	0.		N.		main.tf:3,3-6,4: local.default_tags is declared but not used
Failed	[HIGH] AVD-AZU-0039	0.	0.		N.		Password authentication should be disabled on Azure virtual machines

Summary Code Coverage

Manually run by CG Christos Galanopoulos

View 14 changes

Repositories 2
administration , +1
See Sources card for details

Time started and elapsed
7 Mar at 21:47
3m 40s

Related
0 work items
1 published; 1 consumed

Tests and coverage
Get started

Jobs

Name	Status	Duration
Setup Terraform	Success	31s
Scan Terraform	Success	47s
Deploy Terraform	Success	42s
Create storage container for terraform state file/s	Skipped	
Add repository pipelines	Skipped	

Sources

Repository	Branch / tag	Version	Related
administration Azure Repos	main	52212ac5	None
pipeline-library Azure Repos	main	c2cc42b8	

github-actions bot commented on Feb 12, 2023 • edited

Tfsec Scan Result 🆘 `success`

Terraform Format and Style 🖌 `success`

Terraform Initialization ⚙️ `success`

Terraform Validation 🤖 `success`

▼ Validation Output

```
Success! The configuration is valid.
```

Terraform Plan 📖 `success`

▼ Show Plan

```
Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # azurerm_resource_group.rg will be created
  + resource "azurerm_resource_group" "rg" {
      + id       = (known after apply)
      + location = "westeurope"
      + name     = "rg-tf-dev-weu"
      + tags     = {
          + "environment" = "dev"
          + "owner"       = "christosgalano"
          + "workload"    = "tf"
        }
    }

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

─────────────────────────────────────────────────────────────────────────────

Saved the plan to: development.tfplan

To perform exactly these actions, run the following command to apply:
    terraform apply "development.tfplan"
```

Actor: @christosgalano, Action: `pull_request`, Working Directory: `terraform/environments/development`, Workflow: `deploy`

Tools:

» lint: terraform fmt, tflint, ...

» scan: checkov, trivy, snyk, ...

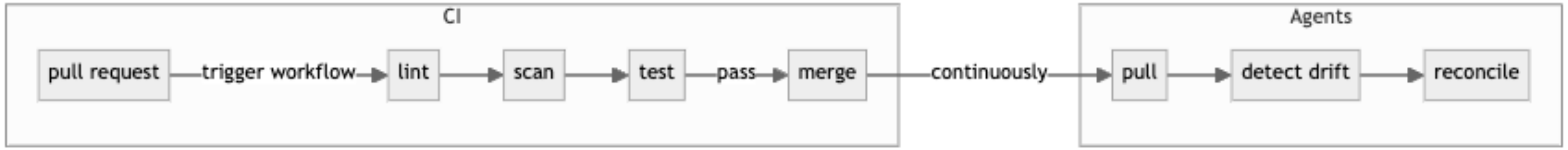
» test: terraform, terratest, kitchen-terraform, ...

» documentation: terraform-docs, ...

» release: semver, ...

Phase **: GitOps

Continuous reconciliation of infrastructure



Tools:

- Flux
- Terraform Cloud
- ArgoCD

Choosing how we organize our Terraform configurations is crucial to building a strong foundation for our infrastructure. As our projects expand and evolve, our code must adapt to support them. Well-organized Terraform code sustains infrastructure evolution and enables us to scale our infrastructure confidently.

Well done is better than well said.

— Benjamin Franklin