Top Ten Terraform Trip-Ups

Microsoft Partner

Gold Cloud Platform
Gold Data Analytics
Gold Data Platform
Gold Application Integration
Silver Project and Portfolio Management

databricks partner

Who Am I?

Name: Ben Somerville Roberts

Job Title: Data Engineer Consultant

Employer: Advancing Analytics

1) Existing Infrastructure Irritation

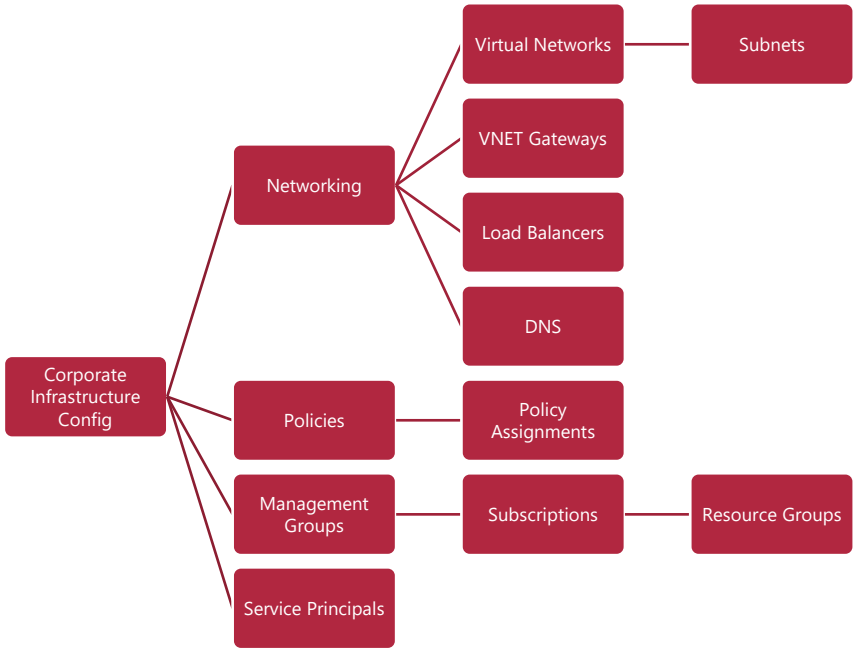
Existing Infrastructure Irritation

It is relatively common to see Terraform or similar tools already in use by centralised Infrastructure teams.

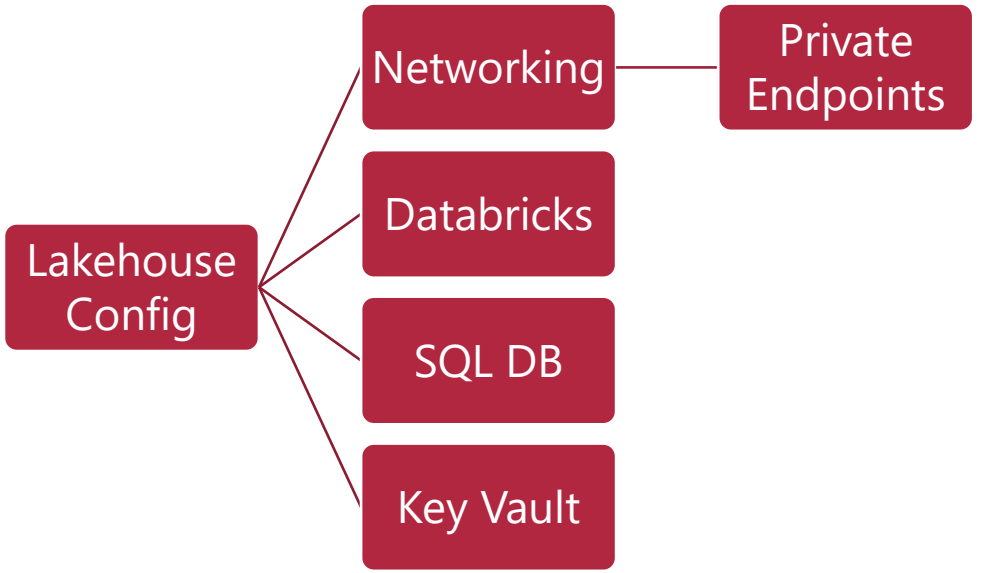
How do ‘client’ Terraform configurations integrate?

Existing Infrastructure Irritation

Infrastructure Team



Data Team



Existing Infrastructure Irritations

Potential Issues:

- Using Remote State to interact means that *everything* in the remote state is exposed
- 'Their' Terraform configuration may differ from 'Yours' and cause changes each time they do a deployment
- Is there a robust process to make sure that 'Their' changes aren't going to break 'Yours'
    - Especially as deployment cadence is likely to be different!

2) Pernicious Permissions

Pernicious Permissions

Q: If I want to carry out the following tasks in Terraform, what permission on the *subscription* do I need?

1) Create a Resource Group
2) Create an ADLS
3) Grant a user access to the ADLS via Azure RBAC

Pernicious Permissions

Q: If I want to carry out the following tasks in Terraform, what permission on the *subscription* do I need?

1) Create an ADLS in an existing Resource Group that Terraform already has an Owner RBAC assignment on
2) Grant a user access to the ADLS via Azure RBAC

Pernicious Permissions

- By asking Terraform to do slightly less, we can drastically reduce the permissions it requires.

- Try to align the permissions given to Terraform to the scope of your project

- Can separate Terraform configurations be used?
  - One to create Resource Groups and assign RBAC roles.
  - One to deploy the data platform

  Does this reduce risk? Can the 'privileged' Terraform configuration be more restricted?

3) Azure AD Agony

Azure AD Agony

Be aware of where your Terraform configuration might have hidden Azure AD Dependencies!

```
resource "azurerm_role_assignment" "probably_not_right" {
  principal_id = "11101010-0101-0101-0101-010101010101"
  scope        = var.key_vault_id
  role_definition_name = "Key Vault Secrets Officer"
}
```

```
module.role_assignments.azurerm_role_assignment.probably_not_right: Still creating... [4m10s elapsed]
module.role_assignments.azurerm_role_assignment.probably_not_right: Still creating... [4m20s elapsed]
module.role_assignments.azurerm_role_assignment.probably_not_right: Still creating... [4m30s elapsed]
module.role_assignments.azurerm_role_assignment.probably_not_right: Still creating... [4m40s elapsed]
module.role_assignments.azurerm_role_assignment.probably_not_right: Still creating... [4m50s elapsed]
module.role_assignments.azurerm_role_assignment.probably_not_right: Still creating... [5m0s elapsed]
module.role_assignments.azurerm_role_assignment.probably_not_right: Still creating... [5m10s elapsed]
```

Azure AD Agony

- Anything that involves translating ‘friendly’ User/Group/Service Principal names into ID values is going to need Azure AD
- Use of AzureAD can be avoided by skipping checks and using ID values instead of display names.

```
vm_admin_group_object_id = "a4f1020c-e1d4-4dce-9f15-b405a0900bc8"
```

Azure AD Agony

Using AzureAD can help reduce your reliance on hardcoded IDs

```
admin_group_name = join("-", ["grp", var.app_name, var.environment, "admin"])
```

```
data "azuread_group" "admin_group" {
  display_name = local.admin_group_name
}
```

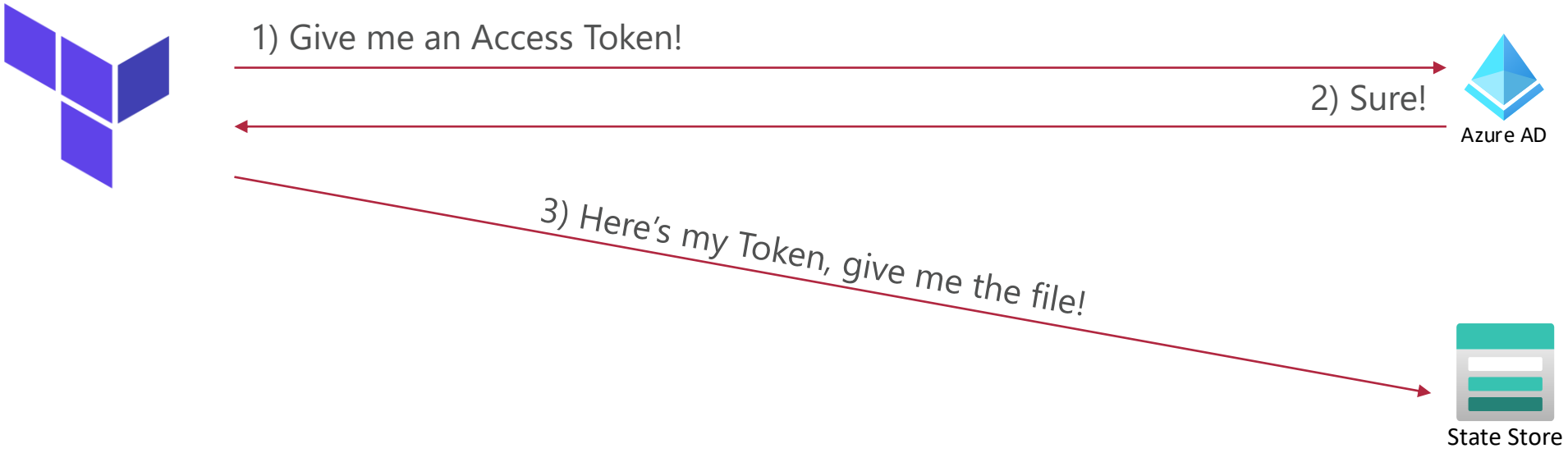
```
resource "azurerm_role_assignment" "demo" {
  principal_id = data.azuread_group.admin_group.object_id
  scope        = data.azurerm_log_analytics_workspace.hub_log_analytics.id
  role_definition_name = "Log Analytics Contributor"
}
```

4) State Store SNAFU

State Store SNAFU

State Files in Azure Storage Accounts with Azure AD Authentication

What you *think* happens:



State Store SNAFU

What actually happens:

1) Give me an Access Token!

2) Sure!

3) Give me the Access Key – Here's my token!

4) Sure!

5) Give me this file – Here's my Access Key!

6) Sure!

State Store SNAFU

Why does this matter?

"Your storage account access keys are similar to a root password for your storage account"

About ACLs

You can associate a security principal with an access level for files and directories. Each association is captured as an entry in an *access control list (ACL)*. Each file and directory in your storage account has an access control list. When a security principal attempts an operation on a file or directory, an ACL check determines whether that security principal (user, group, service principal, or managed identity) has the correct permission level to perform the operation.

Note

ACLs apply only to security principals in the same tenant, and they don't apply to users who use Shared Key or shared access signature (SAS) token authentication. That's because no identity is associated with the caller and therefore security principal permission-based authorization cannot be performed.

Source: https://learn.microsoft.com/en-us/azure/storage/blobs/data-lake-storage-access-control . Viewed 13-MAR-2023

State Store SNAFU

Solution:

- Use a Storage Account per Configuration
  - E.g. Dev, Test, Prod

5) Tenuous Testing

Tenuous Testing

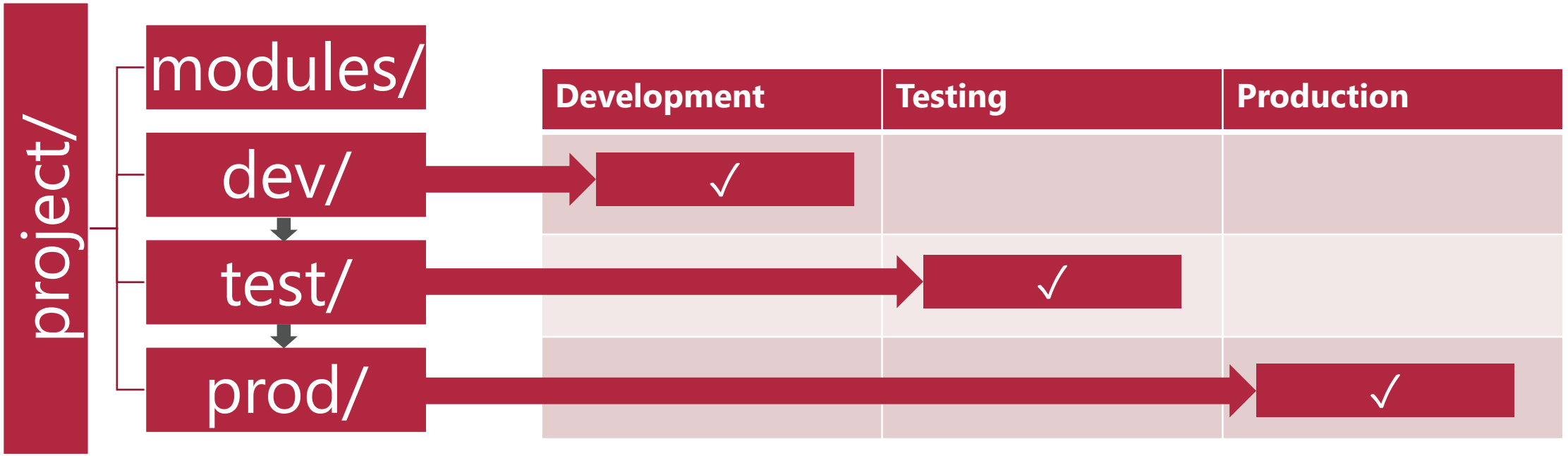
Promotion of 'Normal' code through environments:

Use of tools such as Git and Azure DevOps Pipelines ensure that the same code is deployed through the environments.

There is limited potential for code to be changed during promotion.

Tenuous Testing

Several online guides encourage the use of a “Folder Per Environment” for Terraform projects. Code has to be copy-pasted to promote it!



Tenuous Testing

Potential Problems:

- Changes to modules will immediately take effect across all environments.
- Changes to the environments need to be copy-pasted into new folders
- Production code is never actually tested in it's production form.

Tenuous Testing

Potential Solutions:

- Switch variable files (.tfvars) per environment, not whole configurations
- Use control flags to enable/disable modules per-environment where required

```
resource "azurerm_storage_account" "demo_sa" {
    count = var.create_storage_account ? 1 : 0
    name = "demosa"
    resource_group_name = "demo-rg"
    location = "eastus"
    account_tier = "Standard"
    account_replication_type = "LRS"
}
```

6) Provider Problems

Provider Problems

What Terraform Providers do I need to create and configure a Databricks Workspace?

Create Azure Databricks Workspace → Configure Databricks Workspace

```
provider "azurerm" {
  client_id       = var.client_id
  client_secret   = var.client_secret
  tenant_id       = var.tenant_id
  subscription_id = var.subscription_id
  features {}
}
```

```
provider "databricks" {
  host                        = var.dbx_host
  azure_workspace_resource_id = var.dbx_workspace_id
  azure_client_id             = var.client_id
  azure_client_secret         = var.client_secret
  azure_tenant_id             = var.tenant_id
}
```

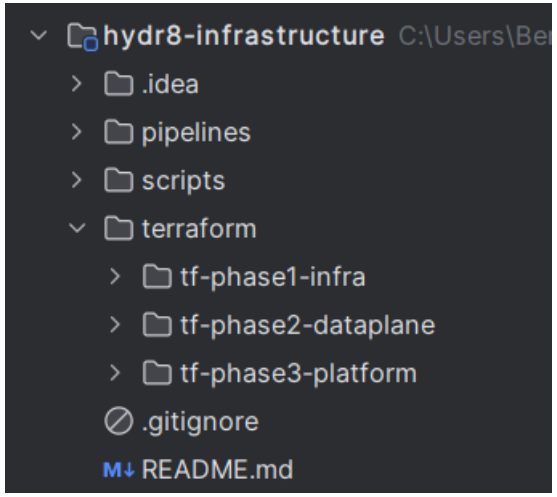
Provider Problems

Problem:

- The Databricks Provider needs the Databricks Workspace info to instantiate
- Terraform instantiates providers on startup
- The Databricks Workspace details aren't available until the AzureRM provider has created it
- If you put it into a module with its own Provider Config, you can't use 'depends_on'
  - This causes issues if you want to use Databricks Provider output in AzureRM resources

Provider Problems

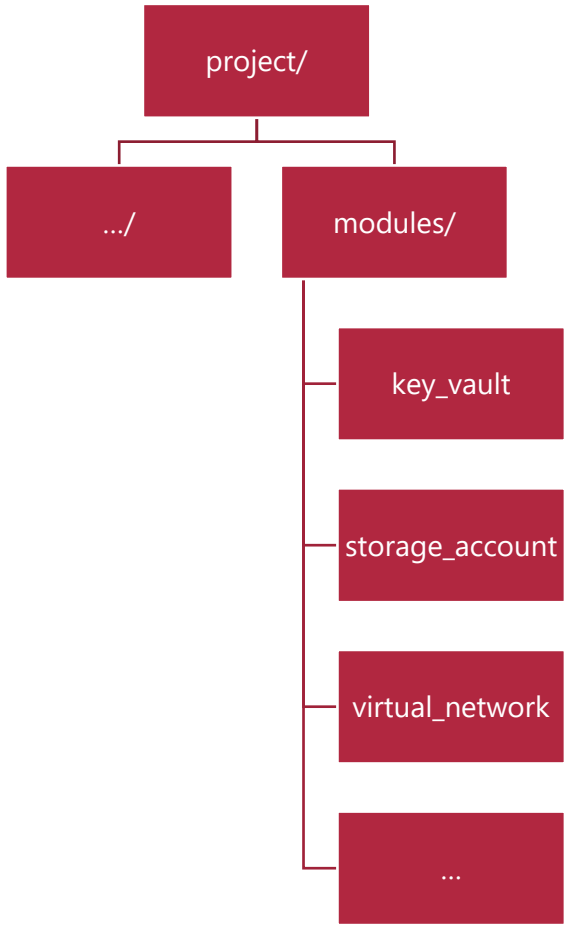
Our Solution:



- Only run the Databricks provider once we're certain the workspace is ready
- Also allows us to accommodate manual activities
- Use of Remote State to pass values between phases

7) Module Malarkey

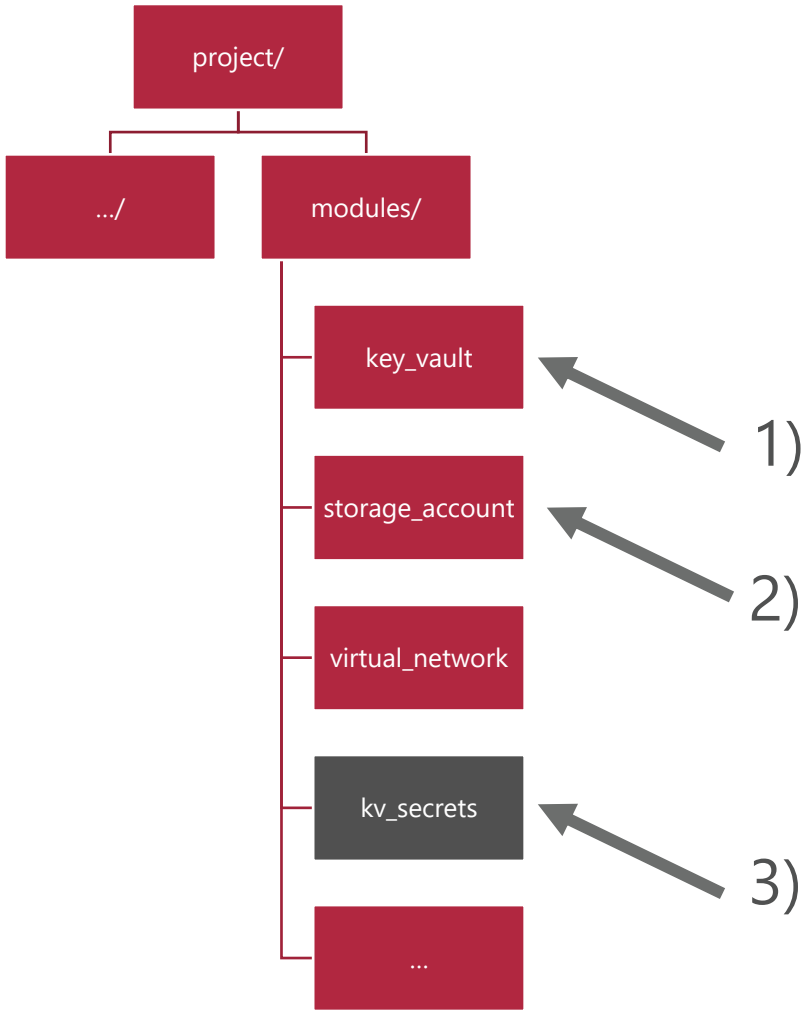
Module Malarkey



Q: If you wanted to store the Storage Account Access Key as a Key Vault Secret, where would you put it?

```
102
103 resource "azurerm_key_vault_secret" "blob_url_secret" {
104   key_vault_id = var.key_vault_id
105   name         = "sa-blob-url"
106   value        = var.blob_url
107   content_type = "text/plain; tfmanaged=true"
108 }
```

Module Malarkey



Option 1 means that the Key Vault module has to 'know' about the Storage Account

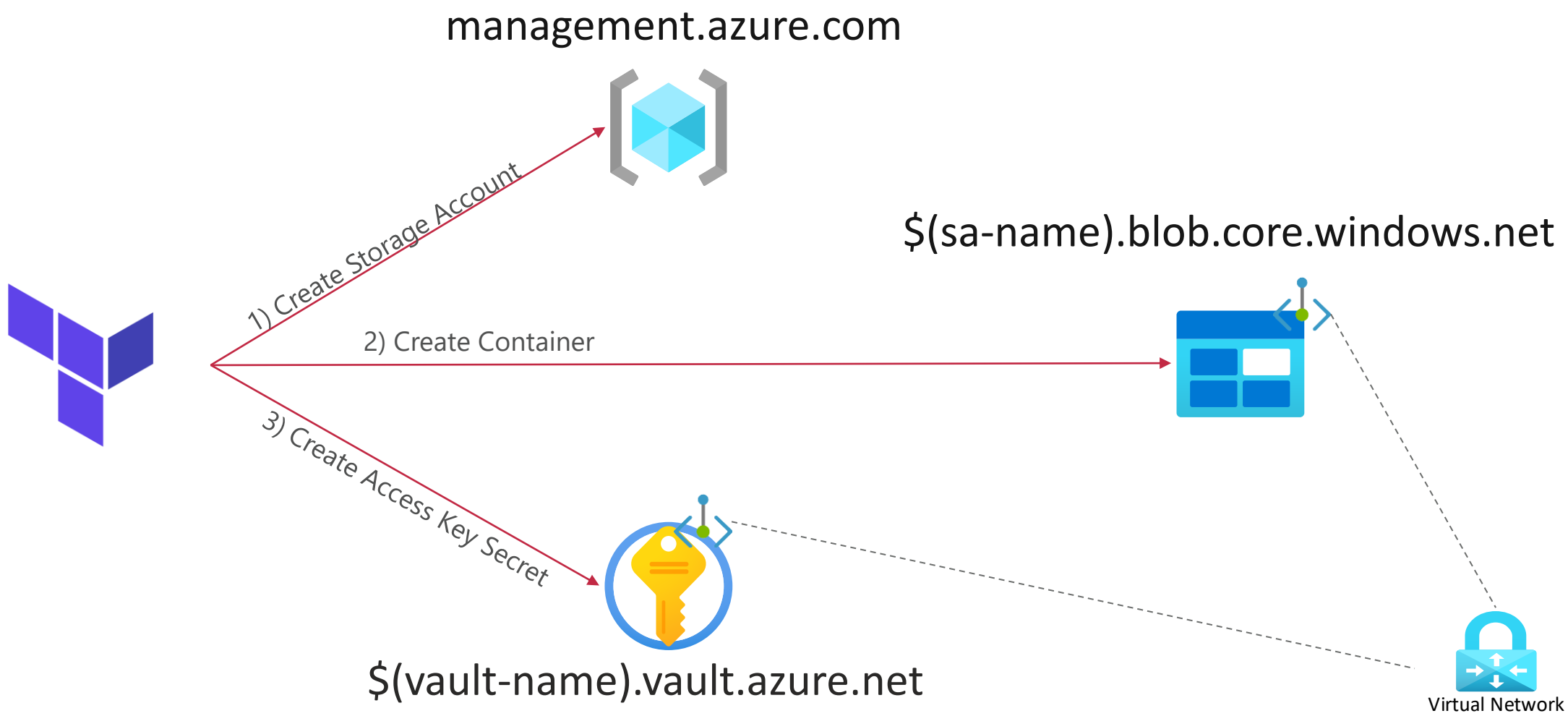
Option 2 means that the Storage Account has to 'know' about the Key Vault

Both options 1 and 2 make it harder to reuse the modules, and make changes more difficult

Option 3 ensures separation of concerns between modules and facilitates reuse

8) Data Plane Disasters

Data Plane Disasters – The Scenario



Data Plane Disasters – The Problem

Some aspects of the AzureRM Terraform Provider carry out operations that are against a resource-specific domain name.

If you have Private Networking configured*, will Terraform be able to access the resources that it has just created?

- DNS
- Firewall
- NSG
- Routes

* As you should!

Data Plane Disasters – The Solution

Potential Solutions:

- Make *everything* managed by Terraform
- Separate anything "Data Plane" related into a separate Terraform configuration
- Accept failures during deployment:
    - Deploy -> Fail -> Manual Tasks -> Deploy -> Success

9) DevOops Pipelines

DevOops Pipelines

Scenario: You've decided to be super-secure. Developers have no access to privileged credentials. All deployments are done via DevOps pipeline.

The build agent is accidentally rebooted half-way through a deployment.

What next?

DevOops Pipelines

If the pipeline is simply restarted, it will fail:

```
C:/ProgramData/chocolatey/bin/terraform.exe apply --var-file=backend_config/azure_provider.tfvars

Error: Error acquiring the state lock

Error message: state blob is already locked
Lock Info:
  ID:        f4a44ea2-800d-7583-72e3-bbe0a20c09d4
  Path:      [redacted].tfstate
  Operation: OperationTypeApply
  Who:       AzureAD\BenSomervilleRoberts@[redacted]
  Version:   1.1.9
  Created:   2023-03-13 16:04:01.5737456 +0000 UTC
  Info:


Terraform acquires a state lock to protect the state from being written
by multiple users at the same time. Please resolve the issue above and try
again. For most commands, you can disable locking with the "-lock=false"
flag, but this is not recommended.
```

DevOops Pipelines

The pipeline is now inoperable. Infrastructure is half-deployed, and you can't force-unlock it from a developer's machine.

DevOops Pipelines

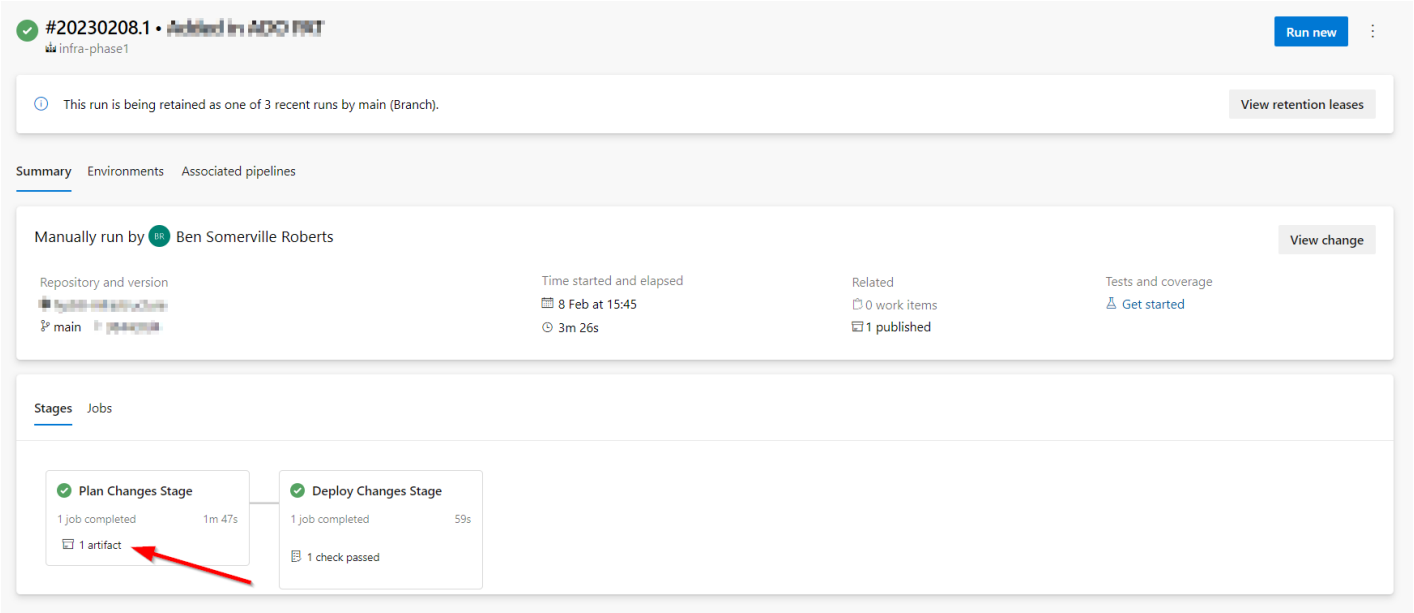
Potential Solutions:

- Build a ‘Force Unlock’ DevOps pipeline to unlock state files
- Procedural controls around deployments and availability of someone who can force-unlock it

10) Awful Artifacts

Awful Artifacts

"So that I can audit what Terraform actions were planned, I will store the Terraform Plan as an Azure DevOps Pipeline Artifact!"



Awful Artifacts

C:\Users\BenSomervilleRoberts\Downloads\out-2263.tfplan\

File Edit View Favorites Tools Help

Add Extract Test Copy Move Delete Info

C:\Users\BenSomervilleRoberts\Downloads\out-2263.tfplan\

Name

- tfconfig
- .terraform.lock.hcl
- tfplan
- tfstate
- tfstate-prev

File Edit Selection View Go Run Terminal Help tfplan - Visual Studio Code

Welcome tfplan

C: > Users > BenSomervilleRoberts > Downloads > tfplan > tfplan

```
34
35    client_secretDC27
36    5��BS"string"�([illegible]DC2:
37    %blob_storage_account_replication_typeDC2DC1
38    SI��BS"string"�LRSDC2?
39    FSadf_vsts_config_account_nameDC2US
40    GS��BS"string"�AdvancinAnalyticsDC2@
41        client_idDC23
42    1��BS"string"[illegible]DC2D
43    FSlog_analytics_workspace_nameDC2$
44    "��BS"string"�log-envemu-dev-loggingDC2F
45    SIsubscription_idDC23
46    1��BS"string"[illegible]DC2%
47    NAKdisable_public_accessDC2FF
48
49    ��ACK"bool"DC2.
50    RSenable_git_integration_for_adfDC2FF
51
```

Awful Artifacts

Potential Solutions:

- Restrict access in Azure DevOps / GitHub Workflows
- Capture the ‘terraform show’ output
- Use a dedicated Terraform visualisation tool.

End.

Feedback!

https://sqlb.it/?10134