Terraform Recommendations

Separate Terraform State Backend Bootstrapping

Problem

You're provisioning the **Terraform state backend using Terraform itself**, creating a **chicken-and-egg problem**.

- You can't run terraform init with a remote backend **until the backend exists**.
- The backend block is **empty or incomplete**, requiring **manual intervention**.
- There's **no separation** between **bootstrapping** and **main infrastructure code**

Solution

Use a **separate, manual/scripted bootstrapping process** outside of Terraform:

- Create the backend manually (or via script):
- Define the backend explicitly in backend.tf:

```
terraform {
  backend "azurerm" {
    resource_group_name  = "rg-tfstate"
    storage_account_name = "tfstatestorage123"
    container_name       = "tfstate"
    key                  = "dev.terraform.tfstate"
  }
}
```

Keep this process **outside your Terraform workflow**, so Terraform isn't dependent on infrastructure it hasn't provisioned yet

Remove Cosmos DB; Use Blob Storage for State Locking

Problem

You're using **Cosmos DB for state locking**, but this is **redundant on Azure**.

- Terraform **natively supports state locking** using Azure Blob Storage lease mechanisms.
- Cosmos DB adds **unnecessary complexity, cost, and operational overhead**

Solution

Remove Cosmos DB from the Terraform state locking setup.
Azure Blob Storage already provides **built-in locking via leases**, which is secure, native, and recommended.

Replace Kops with AKS for Azure Kubernetes

Problem

Your Terraform code outputs a kops create cluster command intended for Azure, but kops is **not fully supported for Azure** — it's primarily built for AWS. This can cause cluster creation failures or unpredictable behaviour

Solution

Replace the kops-based cluster provisioning with **Azure Kubernetes Service (AKS)**, which is the fully managed and supported Kubernetes service on Azure.

Use;

azurerm_kubernetes_cluster

in Terraform for native, stable, and production-grade Kubernetes provisioning on Azure.

Invalid VNet CIDR Block

Problem

You're using **172.0.0.0/16** as the VNet address space, which is **outside the RFC 1918 private IP ranges**.

- This can cause **routing conflicts** and **connectivity issues** with other private networks.
- It violates standards for private IP addressing

Solution

Use **RFC 1918-compliant private IP ranges** for your VNet, such as:

- 10.0.0.0/8
- 172.16.0.0/12
- 192.168.0.0/16

Over-Permissive NSG Rules

Problem

Your Network Security Group (NSG) rules allow **all inbound and outbound traffic from any source to any destination** without restriction.

- This exposes your resources to unnecessary risk from external attacks.
- Violates the principle of least privilege and security best practices

Solution

Restrict NSG rules to allow only **necessary traffic** based on your application requirements, such as:

- Limit inbound traffic to specific trusted IP ranges or subnets.
- Restrict allowed ports to only those needed (e.g., 443 for HTTPS).
- Use explicit deny rules where appropriate.

Example: Replace overly broad rules with specific allow rules targeting known sources and ports to reduce attack surface.

Redundant Terraform State Blob Resource

Problem

The terraform.state blob resource is explicitly created in your Terraform code, but this is redundant because the blob for storing Terraform state is automatically created by Terraform when using Azure Blob Storage backend.

Location: azurerm_storage_blob.state_blob

Solution

Remove the explicit azurerm_storage_blob.state_blob resource from your Terraform configuration to avoid unnecessary resource management and potential conflicts. Let Terraform handle the creation of the state blob automatically during initialization.

Modularity and Code Reuse

Problem:

All infrastructure components (VNet, subnets, NSG, route tables, storage, DNS) are defined in a single module, limiting reusability and maintainability

Solution:

Break out components into independent modules (network, dns, storage, security, etc.) to improve modularity, enable reuse across environments, and simplify code management.