# Backstage Azure VM Template with Terraform Implementation Guide

## Overview

This guide provides a complete approach to create a Backstage software template for deploying simple VMs on Azure using Terraform as Infrastructure as Code. The template uses Backstage's scaffolding system combined with Terraform and GitHub Actions for CI/CD.

## Architecture Components

1. **Backstage Template**: Defines the scaffolding structure and user inputs
2. **Terraform Configuration**: Infrastructure as Code for Azure resources
3. **GitHub Actions**: CI/CD pipeline for deployment automation
4. **Catalog Info**: Service registration in Backstage catalog

## Directory Structure

```
azure-vm-terraform-template/
├── template.yaml                    # Backstage template definition
├── skeleton/                        # Template files to be scaffolded
│   ├── catalog-info.yaml           # Backstage catalog registration
│   ├── .github/
│   │   └── workflows/
│   │       └── terraform-deploy.yml # GitHub Actions workflow
│   ├── terraform/
│   │   ├── main.tf                  # Main Terraform configuration
│   │   ├── variables.tf             # Variable definitions
│   │   ├── outputs.tf               # Output definitions
│   │   └── terraform.tfvars         # Variable values
│   ├── README.md                    # Project documentation
│   └── .gitignore                   # Git ignore patterns
└── docs/
    └── setup-guide.md
```

## Terraform Infrastructure Components

### Core Resources

- **Resource Group**: Container for all resources

- **Virtual Network**: Private network with subnet
- **Network Security Group**: Firewall rules
- **Public IP**: Optional external access
- **Network Interface**: VM network connection
- **Virtual Machine**: Linux or Windows instance

### Advanced Features

- **Conditional Resources**: Public IP only when needed
- **Dynamic Security Rules**: Based on access requirements
- **OS-Specific Configuration**: Ubuntu vs Windows settings
- **Random Password Generation**: For Windows VMs
- **SSH Key Management**: For Linux VMs

## Prerequisites

### Azure Setup

```bash
# Create Service Principal for Terraform
az ad sp create-for-rbac --name "terraform-backstage-deployer" \
  --role contributor \
  --scopes /subscriptions/<subscription-id> \
  --sdk-auth

# Output will be used for GitHub secrets
```

### GitHub Secrets Configuration

Set these secrets in your GitHub repository:

- `ARM_CLIENT_ID`: Service principal client ID
- `ARM_CLIENT_SECRET`: Service principal client secret
- `ARM_SUBSCRIPTION_ID`: Azure subscription ID
- `ARM_TENANT_ID`: Azure tenant ID

### Backstage Configuration

Add to your `app-config.yaml`:

```yaml
catalog:
  locations:
    - type: url
      target: https://github.com/your-org/backstage-templates/blob/main/azure-vm-terraform-temp

integrations:
  github:
    - host: github.com
      token: ${GITHUB_TOKEN}
```

## Implementation Steps

### Step 1: Create Template Structure

1. Create the directory structure shown above
2. Copy all template files to appropriate locations
3. Customize variables and defaults for your organization

### Step 2: Configure Terraform Backend (Recommended)

For production use, set up remote state storage:

```bash
# Create storage account for Terraform state
az group create --name tfstate-rg --location eastus
STORAGE_ACCOUNT_NAME="tfstate$(openssl rand -hex 4)"
az storage account create --name $STORAGE_ACCOUNT_NAME --resource-group tfstate-rg --location e
az storage container create --name tfstate --account-name $STORAGE_ACCOUNT_NAME

# Update main.tf backend configuration with the storage account name
```

Then update the backend block in `skeleton/terraform/main.tf`:

```hcl
terraform {
  backend "azurerm" {
    resource_group_name  = "tfstate-rg"
    storage_account_name = "your-storage-account-name"
    container_name       = "tfstate"
    key                  = "vm-deployments/{{.Values.name}}.terraform.tfstate"
  }
}
```

### Step 3: Test Template Locally

Before deploying to Backstage:

```bash
# Test Terraform configuration
cd skeleton/terraform
terraform init
terraform validate
terraform plan -var-file="terraform.tfvars"
```

### Step 4: Register Template in Backstage

1. Commit template to your repository
2. Add the template URL to Backstage catalog
3. Refresh catalog to see the new template

## Key Features

### User Experience

- **Intuitive Form**: Clear sections for basic info, Azure config, and networking
- **Validation**: Input validation for VM sizes, regions, and naming
- **Conditional Fields**: SSH/RDP options based on requirements
- **Owner Picker**: Integration with Backstage teams and users

### Infrastructure Management

- **Multi-OS Support**: Ubuntu Linux and Windows Server options
- **Flexible Networking**: Optional public IP and configurable security rules

- **Cost Optimization**: Appropriate VM sizes and storage types
- **Security Best Practices**: NSGs, SSH keys, and least privilege access

### DevOps Integration

- **GitOps Workflow**: Infrastructure changes through Git commits
- **Automated Testing**: Format checks and validation in CI/CD
- **Output Management**: VM details and connection info in GitHub
- **State Management**: Remote state storage for team collaboration

## Customization Options

### Adding New VM Sizes

Update `skeleton/terraform/variables.tf`:

```hcl
variable "vm_size" {
  validation {
    condition = contains([
      "Standard_B1s", "Standard_B2s", "Standard_B4ms",
      "Standard_D2s_v3", "Standard_D4s_v3",
      "Standard_E2s_v3", "Standard_F2s_v2"  # Add new sizes
    ], var.vm_size)
    error_message = "VM size must be one of the allowed values."
  }
}
```

### Adding Additional Resources

Extend the Terraform configuration with:

```hcl
# Load Balancer
resource "azurerm_lb" "main" {
  count               = var.enable_load_balancer ? 1 : 0
  name                = "${local.vm_name}-lb"
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name

  frontend_ip_configuration {
    name                 = "PublicIPAddress"
    public_ip_address_id = azurerm_public_ip.main[0].id
  }
}

# Application Gateway
resource "azurerm_application_gateway" "main" {
  count               = var.enable_app_gateway ? 1 : 0
  # Configuration here
}
```

### Environment-Specific Configurations

Create multiple `.tfvars` files:

```bash
# Development environment
terraform/environments/dev.tfvars

# Production environment
terraform/environments/prod.tfvars
```

Update GitHub Actions to use environment-specific variables:

```yaml
- name: Terraform Apply
  run: terraform apply -var-file="environments/${{ github.ref == 'refs/heads/main' && 'prod' ||
```

## Security Considerations

### Network Security

- **Default Deny**: NSGs with explicit allow rules only
- **IP Restrictions**: Configurable source IP ranges
- **Private Networking**: VNet integration and private endpoints
- **Network Segmentation**: Subnet isolation and routing

### Identity and Access

```hcl
# Managed Identity for VM
resource "azurerm_user_assigned_identity" "vm" {
  name                = "${local.vm_name}-identity"
  resource_group_name = azurerm_resource_group.main.name
  location            = azurerm_resource_group.main.location
}

# Assign identity to VM
resource "azurerm_linux_virtual_machine" "main" {
  # ... other configuration

  identity {
    type         = "UserAssigned"
    identity_ids = [azurerm_user_assigned_identity.vm.id]
  }
}
```

### Secrets Management

Integrate with Azure Key Vault:

hcl

```hcl
# Key Vault for secrets
resource "azurerm_key_vault" "main" {
  name                = "${local.vm_name}-kv"
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name
  tenant_id           = data.azurerm_client_config.current.tenant_id
  sku_name            = "standard"
}

# Store admin password in Key Vault
resource "azurerm_key_vault_secret" "vm_password" {
  count        = var.os_type == "windows" ? 1 : 0
  name         = "vm-admin-password"
  value        = random_password.vm_password[0].result
  key_vault_id = azurerm_key_vault.main.id
}
```

## Monitoring and Observability

### Azure Monitor Integration

hcl

```hcl
# Log Analytics Workspace
resource "azurerm_log_analytics_workspace" "main" {
  name                = "${local.vm_name}-logs"
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name
  sku                 = "PerGB2018"
  retention_in_days   = 30
}

# VM Insights
resource "azurerm_virtual_machine_extension" "monitor_agent" {
  name                 = "AzureMonitorLinuxAgent"
  virtual_machine_id   = azurerm_linux_virtual_machine.main[0].id
  publisher            = "Microsoft.Azure.Monitor"
  type                 = "AzureMonitorLinuxAgent"
  type_handler_version = "1.0"
}
```

### Backup Configuration

```hcl
# Recovery Services Vault
resource "azurerm_recovery_services_vault" "main" {
  name                = "${local.vm_name}-vault"
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name
  sku                 = "Standard"
}

# Backup Policy
resource "azurerm_backup_policy_vm" "main" {
  name                = "vm-backup-policy"
  resource_group_name = azurerm_resource_group.main.name
  recovery_vault_name = azurerm_recovery_services_vault.main.name

  backup {
    frequency = "Daily"
    time      = "23:00"
  }

  retention_daily {
    count = 10
  }
}
```

## Troubleshooting Guide

### Common Terraform Issues

1. **Provider Version Conflicts**

bash

```
# Fix with provider version constraints
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.75"
    }
  }
}
```

2. **State Lock Issues**

bash

```
# Force unlock if needed (use cautiously)
terraform force-unlock <lock-id>
```

3. **Authentication Problems**

bash

```
# Verify Azure CLI authentication
az account show
az account set --subscription <subscription-id>
```

## GitHub Actions Debugging

1. **Enable Debug Logging**

yaml

```
env:
  TF_LOG: DEBUG
  ACTIONS_STEP_DEBUG: true
```

2. **Validate Secrets**

```yaml
- name: Validate Azure Auth
  run: |
    echo "Subscription: $ARM_SUBSCRIPTION_ID"
    echo "Tenant: $ARM_TENANT_ID"
    # Don't log sensitive values
```

### VM Access Issues

1. **SSH Connection Problems**

```bash
# Check NSG rules
az network nsg rule list --resource-group rg-vm-name --nsg-name vm-name-nsg

# Verify VM status
az vm get-instance-view --resource-group rg-vm-name --name vm-name --query instanceView.statuse
```

2. **Password Reset (Windows)**

```bash
# Reset VM password
az vm user reset-password --resource-group rg-vm-name --name vm-name --username azureuser --pas
```

## Advanced Configurations

### Multi-Region Deployment

Support deployment across multiple Azure regions:

```hcl
variable "regions" {
  description = "List of Azure regions for deployment"
  type        = list(string)
  default     = ["eastus", "westus2"]
}

# Deploy to multiple regions
module "vm_deployment" {
  for_each = toset(var.regions)
  source   = "./modules/vm"

  location = each.value
  vm_name  = "${var.vm_name}-${each.value}"
  # Other variables...
}
```

### Auto-Scaling Configuration

Add VM Scale Set support:

```hcl
resource "azurerm_linux_virtual_machine_scale_set" "main" {
  count               = var.enable_scaling ? 1 : 0
  name                = "${local.vm_name}-vmss"
  resource_group_name = azurerm_resource_group.main.name
  location            = azurerm_resource_group.main.location
  sku                 = var.vm_size
  instances           = var.instance_count

  # Auto-scaling configuration
  # ...
}
```

## Performance Optimization

### Cost Management

- Use Azure Cost Management integration
- Implement auto-shutdown schedules
- Right-size VMs based on usage metrics

- Use spot instances for non-critical workloads

### Performance Monitoring

- Enable Application Insights
- Configure custom metrics
- Set up alerting rules
- Implement health checks

## Next Steps

1. **Template Validation**: Test with different parameter combinations
2. **Security Review**: Implement organization security policies
3. **Cost Analysis**: Set up cost alerts and budgets
4. **Documentation**: Create user guides and troubleshooting docs
5. **Training**: Provide team training on template usage
6. **Feedback Loop**: Collect user feedback for improvements

## Support and Maintenance

### Regular Updates

- Keep Terraform provider versions current
- Update base VM images regularly
- Review and update security configurations
- Monitor for Azure service deprecations

### Community Resources

- **Terraform Azure Provider**: https://registry.terraform.io/providers/hashicorp/azurerm
- **Azure Documentation**: https://docs.microsoft.com/azure/
- **Backstage Documentation**: https://backstage.io/docs/
- **GitHub Actions**: https://docs.github.com/actions

This comprehensive template provides a solid foundation for VM deployment while maintaining security, scalability, and operational excellence.