**Spin Up, Tear Down, Repeat: Multi-Environment Azure Made Easy**

**Multi-Environment Azure VM Creation Project**

**Problem Statement**

Modern organizations require multiple, isolated environments (such as development, testing, staging, and production) that closely mirror each other in infrastructure but remain logically separated. Manual management of these environments is error-prone, time-consuming, and does not scale. There is a need for an automated, repeatable, and auditable process to provision, update, and destroy cloud infrastructure—specifically Azure Virtual Machines (VMs) and their supporting resources—across multiple environments, while maintaining state and enabling cost estimation.

**When to Use This Solution**

* When you need to manage multiple, similar Azure environments (dev, sit, uat, staging, prod) with consistent infrastructure.
* When you want to automate VM lifecycle management (create, update, delete) for each environment.
* When you require infrastructure-as-code (IaC) practices for versioning, repeatability, and auditability.
* When you need to separate state files and resources for each environment to avoid cross-environment interference.
* When you want to estimate and control infrastructure costs per environment.

**Why This Solution Is Needed**

* **Consistency:** Ensures all environments use the same infrastructure definitions, reducing configuration drift.
* **Automation:** Eliminates manual provisioning, reducing human error and speeding up environment setup.
* **Isolation:** Each environment’s resources and state are separated, minimizing risks of accidental changes across environments.
* **Scalability:** Easily extend or replicate environments as business needs grow.
* **Cost Management:** Built-in cost estimation helps manage and forecast cloud expenses.

**Project Explanation**

**Directory Structure Overview**

multi-env-vm-creation/  
│  
├── scripts/  
│ ├── run.sh  
│ └── backend.sh  
│  
├── environments/  
│ ├── dev.tfvars  
│ ├── sit.tfvars  
│ ├── uat.tfvars  
│ ├── staging.tfvars  
│ └── prod.tfvars  
│  
├── modules/  
│ └── azure-vm/  
│ ├── main.tf  
│ ├── variables.tf  
│ └── outputs.tf  
│  
├── .backend-env  
├── .backend-config  
│  
├── main.tf  
├── variables.tf  
├── outputs.tf  
│  
└── README.md

**Key Components**

* **scripts/**: Contains automation scripts for backend setup (backend.sh, used internally by run.sh) and environment management (run.sh).
* **environments/**: Holds environment-specific variable files (\*.tfvars) for dev, sit, uat, staging, and prod.
* **modules/azure-vm/**: A reusable Terraform module that provisions all Azure resources needed for a VM environment.
* **.backend-env, .backend-config**: Generated files holding backend storage configuration for Terraform state management.
* [**main.tf**](http://main.tf)**,** [**variables.tf**](http://variables.tf)**,** [**outputs.tf**](http://outputs.tf): Root Terraform configuration files that define the provider, variables, outputs, and module usage.

**How It Works**

* The project uses Terraform to define and manage Azure resources.
* Each environment is configured via its own .tfvars file.
* The run.sh script provides a menu-driven interface to manage backend setup, environments, VMs, infrastructure teardown, and cost estimation. It internally calls backend.sh as needed.
* The Azure VM module provisions resource groups, networks, subnets, security groups, public IPs, NICs, and Linux VMs.

**Advantages and Disadvantages**

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| - Consistent, repeatable environment provisioning | - Initial setup requires Azure CLI and permissions |
| - Automated VM lifecycle management | - Scripts are Bash-based; Windows users need WSL or similar |
| - Separation of state and resources per environment | - Manual editing of config files required before first use |
| - Cost estimation built-in | - Cost estimates are approximate, not real-time billing |
| - Easy to extend for more environments or resource types | - All environments must use the same Azure region |
| - Supports full infra teardown per environment or globally | - No built-in support for advanced VM customization (e.g. extensions) |

**Prerequisites**

**Before executing the scripts, ensure:**

* **Azure CLI** is installed and authenticated (az login).
* **Terraform** is installed and available in your PATH.
* You have appropriate permissions in your Azure subscription to create resource groups, storage accounts, and VMs.
* **Manual Updates Required:**
  1. [**backend.sh**](http://backend.sh):
     + Set RESOURCE\_GROUP, LOCATION, STORAGE\_ACCOUNT, and CONTAINER variables.
     + LOCATION must match the location value in all .tfvars files.
  2. [**run.sh**](http://run.sh):
     + Set PROJECT\_PREFIX (e.g., "cbc"). All resource names should start with this prefix.
     + Avoid using - or spaces in backend resource names.
  3. **environments/\*.tfvars**:
     + Each file must have a unique vnet\_cidr for its environment.
     + Each file must have a resource\_prefix (can use - here).
     + location must match the LOCATION in backend.sh.
     + Example:

env\_name = "dev"  
vnet\_cidr = "10.1.0.0/16" # Must be different for each environment  
resource\_prefix = "cbc-dev"  
project\_name = "cbc-multi-env"  
location = "East US"  
admin\_username = "learning"  
admin\_password = "Redhat@12345"  
vm\_numbers = []

* **Consistency:** All naming conventions and locations must be consistent across scripts and tfvars.

**What** [**backend.sh**](http://backend.sh) **Script Does**

* **Purpose:** Sets up Azure backend storage for Terraform remote state management.
* **Actions:**
  + Creates a resource group, storage account, and blob container in Azure.
  + Writes environment variables to .backend-env for use by other scripts.
  + Generates .backend-config for Terraform backend configuration.
  + Outputs backend details for verification.
* **Note:** You do **not** need to run this script manually; run.sh will invoke it automatically if backend resources are missing.

**What** [**run.sh**](http://run.sh) **Script Does**

* **Purpose:** Main automation interface for managing environments and VMs.
* **Features:**
  + Ensures backend is configured before any environment changes (calls backend.sh if needed).
  + Menu-driven interface for:
    - Setting up backend storage.
    - Managing VMs per environment (add, delete, destroy infra).
    - Viewing current VM and backend status.
    - Estimating costs (monthly/daily) for all environments.
    - Deleting backend storage (dangerous operation).
  + Handles workspace selection, tfvars management, and resource naming.
  + Updates .tfvars files as VMs are added/removed.
  + Calls Terraform commands (init, plan, apply, destroy) as needed.
* **Why:** Simplifies and standardizes environment management, reducing manual effort and errors.

**Step-by-Step Implementation**

**1. Prepare and Configure**

* **Update backend.sh:**  
  Set RESOURCE\_GROUP, LOCATION, STORAGE\_ACCOUNT, CONTAINER as per your naming conventions and region.
* **Update run.sh:**  
  Set PROJECT\_PREFIX to a unique value (e.g., "cbc").
* **Update all .tfvars files:**
  + Ensure location matches LOCATION in backend.sh.
  + Each environment must have a unique vnet\_cidr.
  + Set a distinct resource\_prefix for each environment.

**2. Backend Setup**

* **No need to run backend.sh manually.**  
  When you start run.sh, it will check and set up backend storage automatically if required.

**3. Manage Environments and VMs**

* Run the main management script:

./run.sh

* Follow the menu prompts to:
  + Add or remove VMs in any environment.
  + Destroy infrastructure for an environment.
  + View all VM and backend details.
  + Estimate infra costs.
  + Delete backend storage (after all infra is destroyed).

**4. Adding/Removing VMs**

* Select an environment.
* Choose to add or delete VMs; the script updates the corresponding .tfvars and applies changes via Terraform.

**5. Destroying Infrastructure**

* After deleting all VMs in an environment, you can destroy the shared infra (network, resource group, etc.) for that environment.

**6. Deleting Backend Storage**

* Only after all environment infra is destroyed, you may delete the backend storage via the menu (irreversible).

**Conclusion**

This multi-environment Azure VM creation project provides a robust, automated, and scalable solution for managing multiple isolated Azure environments using Terraform and Bash scripting. By leveraging environment-specific configurations and a reusable module, it ensures consistency, reduces manual effort, and supports cost management. The solution is ideal for organizations seeking to streamline their cloud infrastructure operations while maintaining strict environment isolation and repeatability. Proper initial configuration and adherence to naming conventions are critical for successful deployment and management.