Project Guide: Multi-Cloud Terraform Infrastructure

This document provides a comprehensive guide and summary for the multi-cloud infrastructure-as-code project, which manages resources across AWS, Azure, and Google Cloud Platform (GCP) using Terraform.

I. Architectural Overview

The project follows a modular and environment-separated architecture to ensure reusability, consistency, and clear separation between foundational services and application deployments.

Directory	Content	Key Responsibility
modules/	Reusable, cloud-specific, resource definitions (e.g., aws_vm, azure_storage).	**Resource Abstraction** (DRY principle).
base-network/	Foundational network infrastructure (VPC, VNet, Subnets, Security Groups) for all three clouds.	**Network Foundation**.
envs/	Environment-specific configurations (dev, staging, prod) that consume the modules.	**Deployment Logic** and **Environment Isolation**.
.github/workflows/	CI/CD pipeline definition (terraform.yml) using GitHub Actions.	**Automation and Enforcement**.
Root Files	provider.tf, versions.tf, global-variables.tf, backend.tf.	**Global Configuration** and **Project Setup**.

II. Deployment Strategy and CI/CD

The deployment process is separated into two main phases: deploying the network foundation and deploying the application resources. Changes are enforced and automated via GitHub Actions.

A. Phase 1: Deploying the Base Network 🌐

The core networking must be established before application resources can be deployed.

1. **Preparation:** Ensure your cloud provider credentials (AWS, Azure, GCP) are configured locally.
2. **Navigate:** Change directory to ./base-network.
3. **Execute:** Run the standard Terraform workflow:

   ```
   terraform init
   terraform plan
   terraform apply
   ```

4. **Post-Deployment:** Retrieve the network resource IDs (VPC ID, Subnet ID, Security Group ID) from the **base-network/outputs.tf** file. These values **must be updated** as input variables in the corresponding envs/*/main.tf files before proceeding to Phase 2.

B. Phase 2: Deploying Application Environments ⚙️

Application resources (VMs, Storage) are deployed using the environment configurations, which call the standardized modules.

1. **Navigate:** Change directory to the desired environment (e.g., ./envs/dev).
2. **Execute:** Run the standard Terraform workflow:

   ```
   terraform init
   terraform plan
   terraform apply
   ```

3. **Key Configuration Differences:**
   - **dev:** Uses small, low-cost resources (e.g., t2.micro on AWS, LRS on Azure).
   - **staging:** Uses mid-tier, scaled resources to mirror production size and configuration.
   - **prod:** Focuses on high-availability, performance, and robust configuration (e.g., m5.large on AWS, GRS on Azure).

C. CI/CD Automation with GitHub Actions

The **terraform-multicloud/.github/workflows/terraform.yml** file automates infrastructure governance:

Event	Pipeline Action	Outcome and Governance
Pull Request (PR)	Runs **Terraform plan** for all environments (dev, staging, prod).	Plan output is posted as a **comment** on the PR, enabling code review and change impact assessment before merging.
Push to main	Runs **terraform apply -auto-approve** for all environments.	Changes are automatically deployed post-merge. The pipeline uses a **matrix strategy** to apply changes to all environments simultaneously.
Authentication:	Requires **GitHub Secrets** (e.g., AWS_ACCESS_KEY_ID, ARM_CLIENT_SECRET) to be configured in the repository settings to authenticate with the cloud providers.	

III. Configuration and Consistency

A. Global Variables (global-variables.tf)

This file defines project-wide consistency, primarily for resource tagging and naming:

- **business_unit**: Standardized owner for cost tracking.
- **project_name**: Common identifier for cross-cloud resources.
- **common_tags**: A map of tags automatically merged into all deployed resources for billing and operational clarity.

B. Providers and Versions (versions.tf, provider.tf)

- **versions.tf**: Explicitly locks the Terraform core version and provider versions (e.g., aws = "~> 5.0") to prevent unintended state drift or breakage when the code is run in different environments or by different team members.
- **provider.tf**: Configures the connection blocks for **AWS, Azure (azurerm), and Google**

Cloud (google), specifying regions and API versions. Credentials are handled externally via environment variables for security.

C. Remote State Management (backend.tf)

The placeholder configuration in backend.tf serves as a reminder that a **remote backend** (e.g., AWS S3, Azure Storage Account, or Terraform Cloud) is essential for:

1. **State Locking:** Preventing simultaneous updates that could corrupt the state file.
2. **Collaboration:** Allowing multiple team members and the CI/CD pipeline to safely access and modify the state.