# **Azure Well-Architected Framework**

1. **Cost Optimization**

*# qa/qa.tfvars*

vm\_size = "Standard\_B1ls" *# Free-tier eligible*

enable\_spot\_instances = true

*# prod/prod.tfvars*

vm\_size = "Standard\_D2s\_v3" *# Right-sized for production*

reserved\_instances = true

* Use workspace-specific sizing (B-series for QA, D-series for Prod)
* Implement Azure Spot VMs for non-critical workloads
* Tag all resources with cost center metadata
* Set budget alerts in each environment

2. **Operational Excellence**

*# bootstrap/main.tf*

resource "azurerm\_monitor\_diagnostic\_setting" "storage" {

name = "diag-${azurerm\_storage\_account.state.name}"

target\_resource\_id = azurerm\_storage\_account.state.id

storage\_account\_id = azurerm\_storage\_account.state.id

log {

category = "StorageRead"

enabled = true

retention\_policy {

enabled = true

days = 30

}

}

}

* Centralize logs from all environments
* Implement deployment pipelines (GitHub Actions/Azure DevOps)
* Use Terraform workspaces for environment isolation

3. **Performance Efficiency**

*# modules/networking/main.tf*

resource "azurerm\_application\_gateway" "appgw" {

sku {

name = terraform.workspace == "prod" ? "WAF\_v2" : "Standard\_Small"

tier = terraform.workspace == "prod" ? "WAF\_v2" : "Standard"

}

}

* Prod: Premium SKUs with auto-scaling
* QA: Basic SKUs with manual scaling
* Implement CDN for static assets
* Use zone-redundant storage for production

4. **Reliability**

*# prod/main.tf*

module "mysql" {

source = "Azure/mysql/azurerm"

high\_availability = {

mode = "ZoneRedundant"

standby\_availability\_zone = "2"

}

}

*# qa/main.tf*

module "mysql" {

source = "Azure/mysql/azurerm"

high\_availability = {

mode = "Disabled"

}

}

* Prod: Multi-AZ deployment
* QA: Single instance
* Implement backup policies with workspace-specific retention:

backup\_retention\_days = terraform.workspace == "prod" ? 35 : 7

5. **Security**

*# bootstrap/main.tf*

resource "azurerm\_key\_vault" "secrets" {

name = "epam-kv-${local.normalized\_workspace}"

enable\_rbac\_authorization = true

purge\_protection\_enabled = terraform.workspace == "prod" ? true : false

}

*# Environment-specific network rules*

module "network" {

allow\_internet\_access = terraform.workspace == "qa" ? true : false

}

* Prod: Enable purge protection, RBAC-only access
* QA: Relaxed policies for development
* Implement NSG rules per environment:

*# qa allows SSH from anywhere*

*# prod restricts to jumpbox*

Cross-Pillar Implementation Example

*# modules/compute/vmss.tf*

resource "azurerm\_linux\_virtual\_machine\_scale\_set" "example" {

name = "vmss-${var.env}"

instances = var.instance\_count

priority = var.enable\_spot ? "Spot" : "Regular"

eviction\_policy = var.enable\_spot ? "Deallocate" : null

*# Security*

admin\_ssh\_key {

username = "adminuser"

public\_key = file(var.ssh\_key\_path)

}

*# Reliability*

zone\_balance = var.env == "prod" ? true : false

zones = var.env == "prod" ? ["1", "2", "3"] : ["1"]

*# Cost*

tags = {

CostCenter = var.cost\_center

Environment = var.env

AutoShutdown = var.env != "prod" ? "true" : "false"

}

}

Recommended Workflow

1. **Bootstrap** (once):

cd bootstrap && terraform apply

1. **Deploy Environments**:

cd ../prod && terraform init -backend-config=../bootstrap/backend.prod.hcl

terraform apply -var-file=prod.tfvars

Monitoring Setup

Add to your bootstrap:

resource "azurerm\_monitor\_action\_group" "alerts" {

for\_each = toset(["qa", "prod"])

name = "CriticalAlerts-${each.key}"

resource\_group\_name = azurerm\_resource\_group.state.name

short\_name = "crital-${each.key}"

email\_receiver {

name = "admin-team"

email\_address = each.key == "prod" ? "prod-alerts@company.com" : "dev-alerts@company.com"

}

}

This implementation gives you:

* Cost visibility through tagging and sizing
* Operational consistency through IaC
* Performance differentiation between environments
* Reliability appropriate to each stage
* Security controls scaled by environment risk

**FrontEnd**

* Para desplegar el frontend en Azure existen varias opciones tales como:
* Static Web pages
* **App Services**
* Container Instances (Kubernetes)
* Virtual Machines
* Blob Storage

*Se selecciona App Services porque es un servicio administrado con soporte para Node.js, tiene escalado automático, tiene integración para CI/CD, tiene opciones gratuitas y de pago (F1).*

* Se usará un **NAT Gateway** ya que es un servicio totalmente administrado que permite a los recursos en una red virtual (VNet) acceder a Internet de manera segura y escalable **sin necesidad de una dirección IP pública** asignada directamente a cada recurso.
* Se usa App Services integrado a una VNET, porque se requiere acceder a una API y BD interna.