**🚀 Overview**

This repository contains a Terraform-based setup for managing Vault secrets and Docker containers for multiple microservices (account, gateway, payment, and frontend) across three environments: development, staging, and production.

**🧠 Design Decisions**

**1. Provider Configuration**

* Separate Vault providers are defined for each environment to isolate secrets and policies.

**2. Modularization Strategy (WIP)**

* The code is currently repetitive, but we plan to:
  + Create a reusable module for Vault secret, policy, and user provisioning per service.
  + Create a second module for Docker container deployment.
  + Use for\_each with a variable map to simplify environment/service expansion.

**3. Environment Isolation**

* Resources are clearly namespaced by environment (e.g., account\_development, account\_staging, account\_production).

**4. Lifecycle Configuration**

* Docker containers use ignore\_changes = all to avoid recreation on minor config drift.

**5. Explicit Passwords**

* For demonstration, passwords are hardcoded in secrets. In production, we'd use Vault dynamic secrets or reference from a secure password store.

**Note:** I am proficient with Terraform. However, in this repo I’ve focused more on clarity and extensibility than full optimization or module reuse (which will be the next step).

**🔄 CI/CD Integration**

This Terraform code can be integrated into a CI/CD pipeline with tools like:

**Option 1: GitHub Actions**

name: Terraform Plan & Apply

on:

push:

branches: [main]

jobs:

terraform:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- uses: hashicorp/setup-terraform@v2

with:

terraform\_version: 1.6.0

- run: terraform init

- run: terraform validate

- run: terraform plan

**Option 2: Jenkins or GitLab CI**

* Use Docker or a Terraform runner image
* Securely inject Vault tokens and environment-specific variables

**🌐 Production Considerations**

When using this code in a real-world scenario, we would consider the following enhancements:

**✅ Security & Secrets Management**

* Replace hardcoded Vault tokens and passwords with dynamic secrets and Vault AppRole authentication.
* Use terraform-vault-provider with token renewal logic.

**🔒 Role-Based Access**

* Apply principle of least privilege in Vault policies.

**🧪 Testing**

* Introduce automated integration testing using terratest or kitchen-terraform.

**🧱 State Management**

* Use remote backends (e.g., Terraform Cloud, S3 + DynamoDB) to manage state securely.

**🏗️ Modular Design**

* Fully implement modules for Vault resource provisioning and container orchestration.
* Simplify the addition of new services and environments.

**🛑 Drift Detection & Monitoring**

* Periodically run terraform plan in CI to detect drifts.
* Monitor container health and Vault logs.

**✅ Summary**

This solution establishes a foundation for secure secret management and service provisioning with Terraform. It’s designed for clarity and ease of expansion, with future steps targeting modularization and CI/CD automation.