Vistra GEP DevOps Engineer Technical Assignment

Serverless Infrastructure Design with Terraform and AWS

Assignment Overview

Design and document a serverless REST API application using Terraform for infrastructure provisioning and GitHub Actions for CI/CD automation. You will create production-ready Infrastructure as Code that could be deployed to AWS, with validation done through code review, Terraform validation, and documentation.

The assignment includes designing your own project organization based on best practices. This assessment tests understanding of Infrastructure as Code principles, DevOps patterns, and engineering judgment.

Maximum Duration: 1 week

Technology Stack

- **Version Control**: GitHub
- **Infrastructure as Code**: Terraform (v1.5+)
- **Target Cloud Platform**: AWS (Lambda, API Gateway, DynamoDB, EventBridge, CloudWatch)
- **CI/CD**: GitHub Actions (validation workflows only)
- **Programming Language**: Node.js 22.x for Lambda functions

What You Will NOT Need

- AWS Account or credentials
- LocalStack or local AWS emulation
- Actual resource deployment
- Cloud costs or infrastructure provisioning

What You WILL Do

- Design your own project folder structure based on best practices
- Write production-ready Terraform code
- Create Node.js 22 Lambda function code
- Design CI/CD validation pipelines
- Document architecture and design decisions
- Justify your organizational choices
- Use `terraform validate` and `terraform fmt` for code quality

Prerequisites

Install the following tools on your local machine:

1. **Terraform CLI** (v1.6 or higher)
2. **Node.js 22.x**
3. **Git**
4. **Code Editor** (VS Code recommended)

Verify installations:

```
terraform version  # Should show v1.5+
node --version     # Should show v22.x
git --version
```

Core Tasks (Required)

Task 1: Project Setup & Infrastructure Foundation

Design and implement your Terraform project structure for a serverless REST API application. Organize your project to reflect Infrastructure as Code best practices, considering maintainability, scalability, and clarity for team collaboration.

Implement Terraform code for core infrastructure components:

- S3 bucket for Lambda deployment packages with versioning enabled
- DynamoDB table for application data with encryption enabled
- IAM roles for Lambda execution following least-privilege principles
- CloudWatch Log groups for structured logging
- OPTIONAL: Write tests for Terraform code

Deliverables:

- Complete project repository with your chosen organization
- Terraform code following HashiCorp conventions
- Clear README explaining your folder structure and the reasoning behind your organizational approach
- Terraform configuration files with proper variable definitions and validation rules
- Outputs file containing important resource identifiers and endpoints

Task 2: Serverless API with Lambda & API Gateway

Implement production-ready Lambda functions and API Gateway infrastructure.

Lambda Functions (Node.js 22):

- Implement CRUD API handlers for items management:
  - POST /items - Create item
  - GET /items - List items
  - GET /items/{id} - Get single item
  - PUT /items/{id} - Update item
  - DELETE /items/{id} - Delete item
- Use ES Module syntax
- Implement proper error handling with appropriate HTTP status codes
- Include input validation for all requests
- Implement structured JSON logging
- Use AWS SDK v3 with modular imports
- DO NOT need to implement actual CRUD code that makes changes to database

Infrastructure:

- Lambda function deployment configuration
- API Gateway REST API with proxy integration
- CORS configuration
- DynamoDB table integration
- IAM permissions following least-privilege principles

Deliverables:

- Node.js 22 Lambda function code using ES Modules
- Terraform modules for Lambda, API Gateway, and DynamoDB
- Complete Terraform configuration with all necessary resources
- Documentation in Lambda functions directory explaining the implementation
- Outputs file showing API endpoint and resource identifiers

Task 3: CI/CD Pipeline with GitHub Actions

Implement validation and security scanning workflows.

Workflows should:

- Validate Terraform code on every PR and push:
  - Run `terraform fmt -check` to verify formatting
  - Run `terraform init -backend=false` (initialize without backend)
  - Run `terraform validate` to check syntax and configuration
  - Provide feedback on validation results

- Validate Node.js Lambda functions:
    - Setup Node.js 22
    - Install dependencies
    - Build Lambda function
    - Package for deployment
- Include security scanning for Terraform code (tfsec, Checkov, or similar)
- Include documentation validation (markdown linting)

Deliverables:

- GitHub Actions workflow files in `.github/workflows/`
- All workflows execute successfully without AWS credentials
- Documentation explaining the CI/CD strategy and how the pipelines support code quality

Advanced Tasks (Optional - Choose One or Both)

Task 4: Monitoring & Observability

Design monitoring infrastructure for production readiness.

Implement:

- Terraform configuration for CloudWatch dashboards
- Alarms for Lambda errors, latency, and throttles
- Dashboards for API Gateway metrics
- DynamoDB monitoring configuration
- Structured logging configuration in Lambda functions
- Documentation of monitoring strategy and alert thresholds

Deliverables:

- Terraform configuration for monitoring resources
- Documentation of monitoring approach
- Explanation of metric selection and alert thresholds
- Example CloudWatch Insights queries for common troubleshooting scenarios

Task 5: Event-Driven Architecture

Design event-driven patterns using EventBridge.

Implement:

- EventBridge rule configuration

- Scheduled Lambda function for background processing
- Dead-letter queue for failed events
- Event-driven Lambda function in Node.js 22
- Integration with DynamoDB

Deliverables:

- Terraform configuration for EventBridge resources
- Event processor Lambda function using Node.js 22 and ES Modules
- DLQ configuration and monitoring
- Documentation of event flow and error handling strategy
- Architecture diagram showing event-driven patterns

Evaluation Criteria

Project Organization & Architecture (30%)

Your folder structure should reflect best practices in Infrastructure as Code and DevOps. The organization will be evaluated based on logical separation of concerns, modularity, scalability, and the clarity of your rationale. Well-organized code demonstrates architectural thinking and makes maintenance and extension easier for teams.

Technical Execution (35%)

Terraform code should follow HashiCorp best practices and conventions. Node.js 22 functions should use ES Modules and modern async patterns. IAM policies should follow least-privilege principles. Code should be clean, readable, and well-structured. Variables should include validation rules and helpful descriptions.

DevOps Practices (20%)

CI/CD workflows should provide appropriate automation and validation. Infrastructure as Code should follow standards for formatting, documentation, and outputs. Security should be addressed through automated scanning and proper secrets handling. Your implementation should demonstrate solid DevOps engineering judgment.

Documentation & Communication (15%)

Your README and supporting documentation should clearly explain the design, architecture, and your decision-making process. Architecture diagrams should help visualize the system. The rationale for your organizational and technical choices should be evident to reviewers.

Submission Guidelines

What to Submit

1. GitHub Repository (public or grant reviewer access)
2. README.md at repository root containing:
    - Project overview
    - Your folder structure explanation and rationale
    - Key architectural decisions made
    - Instructions for validating the code locally
    - Any assumptions or trade-offs considered
3. Architecture diagram (ASCII art, Mermaid, or image file)
4. Documentation explaining your project structure and design decisions

Repository Requirements

- All Terraform code passes `terraform validate` (no AWS account needed)
- All Terraform code formatted with `terraform fmt`
- Node.js 22 used for all Lambda functions
- ES Module syntax used throughout Lambda functions
- GitHub Actions workflows present and properly configured
- No AWS credentials or secrets committed
- Clear README explaining your approach
- Project organization demonstrates DevOps best practices
- Architecture diagram included in documentation

Local Validation

Create a validation approach to demonstrate that your code meets the requirements. This might include:

- Script to run `terraform validate` and `terraform fmt -check`
- Node.js build and packaging validation
- Verification that all code follows conventions

All validation should work without AWS credentials or cloud account access.

Helpful Resources

Terraform Documentation and Best Practices

- HashiCorp: Standard Module Structure
- HashiCorp: Best Practices for General Style and Structure

- AWS: Best Practices for Code Base Structure and Organization
- Terraform Best Practices: Code Structure

Project Organization

- Terraform Files and Project Structure
- AWS: Best Practices for Organizing Larger Serverless Applications
- Organizing Serverless Projects guides

Node.js 22 and AWS Lambda

- Node.js 22 AWS Lambda Runtime documentation
- ES Modules in Lambda
- AWS SDK for JavaScript v3

GitHub Actions and CI/CD

- GitHub Actions documentation
- Terraform CI/CD and testing patterns

Notes

This assignment evaluates your engineering judgment through the folder structure you design, the way you organize modules, and how you document your decisions. These organizational choices communicate your experience level as clearly as the code itself.

The assignment does not require deploying to AWS. All validation should be achievable locally using Terraform's built-in validation tools. Focus on creating code that demonstrates production-ready practices and thoughtful infrastructure design.

There is no single "correct" folder structure. Different approaches have different trade-offs. Your ability to explain and justify your choices is as important as the choices themselves.