COALFIRE

AWS Technical Challenge

Welcome to Our Technical Challenge!

I'm glad you're here and excited to see what you bring to the table.
Our goal is to make the hiring process a true reflection of your skills while respecting your time. Rather than multiple technical interviews or live coding sessions, this take-home challenge lets you work at your own pace, in your own environment, and show us what you can really do.

Here's the deal:

- We expect this challenge to take a few hours If you want to dive deeper, explore more features, or fine-tune your solution, feel free, but it's not expected. We value work-life balance, and we're not here to overwhelm you.
- This challenge is meant to give you a representation of what we actually work on. No trick questions, no gotchas, just a real-world problem you can dig into.

What we're looking for:

- How you think through and solve problems.
- Your technical skills, including code quality and approach.
- Communication and clarity in how you present your work, whether in code or supporting notes.

When you're done, send us your GitHub Repo so we can dive into your solution and learn more about how you work.
If you run into any questions or need anything clarified along the way, don't hesitate to reach out.
We can't wait to see what you come up with!

Douglas Francis
Cloud Services Managing Principal

Challenge Overview

Part One: Deployable Infrastructure

Create a proof-of-concept AWS environment using Terraform. The environment will host a basic web server with proper network segmentation and security controls.

You must use Terraform modules (your own or any open source). Coalfire's modules are encouraged but optional. (https://github.com/orgs/Coalfire-CF/repositories?q=visibility:public+terraform-aws)

Your challenge when submitted should be all within a public GitHub repository, with your code, a diagram that depicts your solution and any notes you have from going through the challenge. The main repo README should cover your solution, provide deployment steps and your notes and commentary from going through the challenge.

Any detail not provided in the scenario or requirements is up to your discretion.

Technical requirements

- 1 VPC – 10.1.0.0/16
- 4 subnets (spread evenly across two availability zones)
  - Sub1 – 10.1.0.0/24 (should be accessible from internet)
  - Sub2 – 10.1.1.0/24 (should be accessible from internet)
  - Sub3 – 10.1.2.0/24 (should NOT be accessible from internet)
  - Sub4 – 10.1.3.0/24 (should NOT be accessible from internet)

- 1 EC2 instance running Red Hat Linux in subnet sub2
  - 20 GB storage
  - t2.micro
- 1 auto scaling group (ASG) that will spread out instances across subnets sub3 and sub4
  - Use Red Hat Linux
  - 20 GB storage
  - Script the installation of Apache web server (httpd) on these instances
  - Add an IAM role to your ASG hosts that can read from the "images" bucket
  - 2 minimum, 6 maximum hosts
  - t2.micro
- 1 application load balancer (ALB) that listens on TCP port 80 (HTTP) and forwards traffic to the ASG in subnets sub3 and sub4 on port 443
- Security groups should be used to allow necessary traffic
- An IAM role that can write to the logs to log bucket from ALL EC2s provisioned.
- 1 S3 bucket: “Images” with a folder called archive
  - “Memes” folder - move objects older than 90 days to glacier.
- 1 S3 bucket: “Logs” with two folders and the following lifecycle policies
  - “Active folder” - move objects older than 90 days to glacier.
  - “Inactive folder” - delete objects older than 90 days.

Part Two – Documentation

- GitHub README with
  - Solution overview
  - Deployment instructions
  - Design decisions and assumptions
  - References to resources used
  - Assumptions made
  - Improvement plan with priorities
  - Analysis of operational gaps
  - Evidence of successful deployment against the criteria (e.g., screenshots, CLI output, Terraform apply logs)
  - Solutions Diagram embedded
- Solutions Diagram with
  - Properly formatted and styled diagram against AWS diagram standards
  - All deployed components identified
  - Connectivity/dataflow’s identified

Deliverables

A link to your Public GitHub repository containing:

- All Terraform configurations
- Solutions diagram
- Full README

Evaluation Criteria

We evaluate tech challenges based on

- Code Quality
  - Terraform best practices
  - Module usage
  - Correct resources deployed
  - Clear, maintainable code
- Architecture Design
  - Diagram clarity
  - Resource organization
- Documentation
  - Clear instructions
  - Well-documented assumptions
  - Proper reference citations
- Problem-Solving Approach
  - Solutions to challenges encountered
  - Design decisions
  - Comments about challenges presented and how you approached them

Guidelines

- Work independently – no collaboration
- We do encourage use of web resources (Stack Overflow, Reddit, technical blogs, etc.) if used provide links as part of your documentation.

- Document any assumptions and design decisions you make.
- Be realistic about scope. Partial solutions are fine if well documented.
- Questions welcome – reach out if you need clarification