Arqiva Technical Challenge Senior DevOps Engineer

Lewis Desforges

My response to this challenge, including source code, can be found in my Github repository:
https://github.com/gitLRD/arqiva-technical-challenge.git

Being given a light brief and a small amount of time for this challenge, my guiding principle was to keep it simple, lightweight, and not make many assumptions.

Current solution

Hosting

I chose to host this solution in AWS as it's the cloud provider I am most familiar with. There are several reasonable ways to serve webpages in AWS, but I chose an S3 bucket because it's serverless.

Using S3 means there are no servers or containers to maintain, which saves time on building and maintaining the solution, and reduces cost when there is no demand on the system (due to a lack of idling servers). It also gives access to AWS' gamut of resources without need to modify, so this small scale test could be shared with thousands of users without needing to consider storage throughput or CPU capacity.

Infrastructure as Code

The infrastructure is coded in Terraform, because I know it best, but it has the additional benefit of being cloud provider agnostic.

The Terraform code consists of these items of note:

Resource: aws_s3_bucket	The storage container the HTML file (index.html) will be kept in
Resource: aws_s3_bucket_policy	The access policy for the S3 bucket. Best practice is usually to restrict access to only those who need it (least privilege) but demonstrating the solution would likely be done outside the AWS network, so narrowing access to a principle would have been very difficult.
Resource: aws_s3_object	The HTML file itself. Given the brevity of the file I decided to define its content inline for ease and keeping the file structure tidy, but if the file was larger I would have kept it as a separate file and referenced its source in the Terraform.
Var: arqiva-dynamic-string	The brief stated that the dynamic string should be "set to whatever is requested", which suggests that we want to be able to change it quickly but be persistent. I decided to make the dynamic string part of the HTML file and adjust it on deployment using Terraform. This required the use of a variable.

Future improvements

This system is very basic but suits the brief. Given a more complex system I would make the following changes:

1. Store the Terraform state file in an S3 bucket. This is a minor and almost meta improvement but would allow multiple developers to deploy Terraform code with a consistent state file.
2. Tag infrastructure components to make it easier to identify which components belong to which system. This would also make understanding the AWS bill easier.
3. If the infrastructure became more complex it would be wise to implement some automated testing to validate the Terraform code. These could be tied in with pull requests to make sure that only valid/working infrastructure gets merged to the master branch. GitHub Actions or web hooks would be the best way to run these tests with each pull request and keep a record of the test results.
4. If a more complex website is required, like one using a CMS solution, the site could be hosted from an EC2 instance or ECS container and be linked to an AWS RDS.