Technical test

Context

The client requested a product that takes a parameter and returns a HTML string with the format:

```
<h1>The saved string is <parameter></h1>
```

This parameter must be received on runtime, and can be changed without redeploying the solution. Also, the whole infrastructure must be deployed using an automated solution. This solution must be idempotent, so every user that uses the same parameter and the same version, should receive the same answer. Also it was requested that the solution delivers the string within a Web page.

For this solution, I was trying to deliver the simplest and most cost-effective approach. The analysis was focused on 2 aspects:

- The software
- The infrastructure that allows the solution to work

And this should be delivered as fast as possible.

The software

Since the requirements were focused on a web environment, I considered these options for the software:

- NodeJS
- Python
- Perl

I decided to use Python, because I am more familiar with it and could deliver a solution faster than using NodeJS. Perl was discarded because it would need a more complicated and more expensive infrastructure.

The infrastructure

For delivering this solution, I chose AWS for hosting the infrastructure. This was the cheapest option because we will be using the free tier, and also the position is mainly for AWS. On AWS, we could deliver this solution using:

- EC2
- EKS Fargate
- ECS Fargate
- AWS Lambda

Comparison:

Solution	Favour	Against	Approximate Monthly Costs
EC2	1. Full configurable VM 2. No need for an Application Load Balancer (ALB)	1. Must configure and install software manually 2. Must use another solution (i.e. Ansible) for automate the installation 3. Harder to maintain because more pieces to administer 4. For scaling the solution you must add pieces	10 USD (for single machine t3.nano)
EKS Fargate	1. Based on a proven solution in the market (kubernetes) 2. Can automatically scale based on usage	1. Must build the docker image 2. Must configure ingress and DNS 3. Must configure the Kubernetes manifests 4. Harder to implement Continuous Delivery using Terraform 5. Ideally is to use a Private Registry within AWS (extra cost)	US$115 with Application Load Balancer (ALB)
ECS Fargate	1. Based on a	1. Must build	US$42 with

	solution with dockers 2. Can automatically scale based on usage 3. Totally serverless	the docker image 2. Must configure ingress and DNS 3. Must configure the Services and Tasks in ECS 4. Ideally is to use a Private Registry within AWS (extra cost) 5. Less available tools that EKS	Application Load Balancer (ALB)
AWS Lambda	1. Can use a simple script or a docker image 2. Charge based on usage 3. Automatic scale 4. Can use Terraform for the whole cycle.	1. Ideally for short and quick process 2. Can be expensive than other solutions over heavy loads	US$2 (With API Gateway)

For a solution that will not have a heavy load, that can run with a limited amount of resources, and an operation simple and not time demanding, like the one requested, AWS Lambda with API Gateway seemed a good fit. Also, looking for the financial scenario, AWS Lambda with API Gateway is the best solution for this infrastructure.

Script deployed

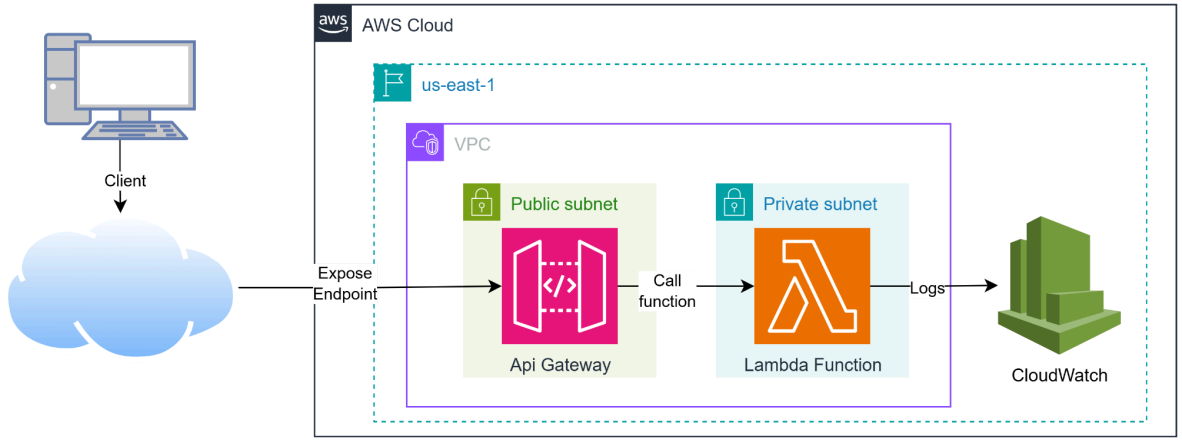
The script deployed has 3 functions:

1. handler
   This function loads the parameter from the querystring, call the other 2 functions, and returns the message
2. inputValidator
   This function makes input sanitation. It receives a string and checks for non-empty string, and/or non-alphanumeric characters.
3. htmlBuilder

This function builds the webpage, and inserts the prepared string based on the input

Infrastructure

The infrastructure for this solution is as follows:



One VPC with 2 subnets (one public, one private). The Public subnet has the implementation of the API Gateway, which handles the public endpoint. In the private subnet, is the AWS Lambda function, isolated to provide more security. All logs are stored in the CloudWatch services.

The public subnet has one security group that allows connection from every client to the https port (443). The private subnet has one security group that allows connection only from machines in the public subnet. Both groups have an egress rule that allows all outbound traffic.

Terraform

The Terraform configuration is divided into files. The name of file describes what the script create:

Name	Description
main.tf	Provider configuration
networking.tf	VPC and subnetwork creation
security.tf	Security groups and rules
AWS Lambda.tf	AWS Lambda and API Gateway management

Also, the Terraform configuration handles variables, that can be used for an easy configuration of the script

Variable name	Description	Default value
region	AWS region	us-east-1
access_key	AWS access Key	
secret_key	AWS secret Key	
vpc_cidr	VPC CIDR Block	10.1.0.0/16
subnet_public_cidr	Subnet CIDR Block	10.1.1.0/24
subnet_private_cidr	Subnet CIDR Block	10.1.2.0/24

Further improvement

At this point, I would recommend the following improvements:

- Have a template for the answer page, that can be stored on a S3 bucket, and read by the AWS Lambda function. The template can be passed as a parameter to the function. Using this I can work with the design and UX without changing the code.
- Use a Terraform backend (at least a S3 bucket) so the state can be shared across several teams, and the platform can share the state.
- Use a CI/CD pipeline to run unit tests (added in the repository, in the file unit_test.py) and make Terraform run automatically.
- Run a little stress test, in order to improve and fine-tuning AWS Lambda configuration.