HOSTING A STATIC SITE ON AWS: IS CLOUDFRONT ALWAYS THE RIGHT CHOICE?

Amazon CloudFront | Amazon S3 | Infrastructure as Code (IaC)

beSharp | 26 June 2020

Introduction

Binomial services **CloudFront** and **S3** are nowadays part of consolidated practices for many companies that have the necessity of hosting a static website (or part of it) while **keeping its costs low without sacrificing security standards** (eg: SSL termination on proprietary domain). But is this the only way pursuable? Are there any other means by which we can achieve the same result by also solving other necessities? This article tries to give an answer to that!

Problem

Imagine you have to handle the following need: your company's frontend team requests testing a freshly developed feature on a production-like infrastructure to show and verify that the result is somewhat the one expected for the release. In particular, it would like to have a different domain name for every feature developed in order to avoid path related inconsistencies.

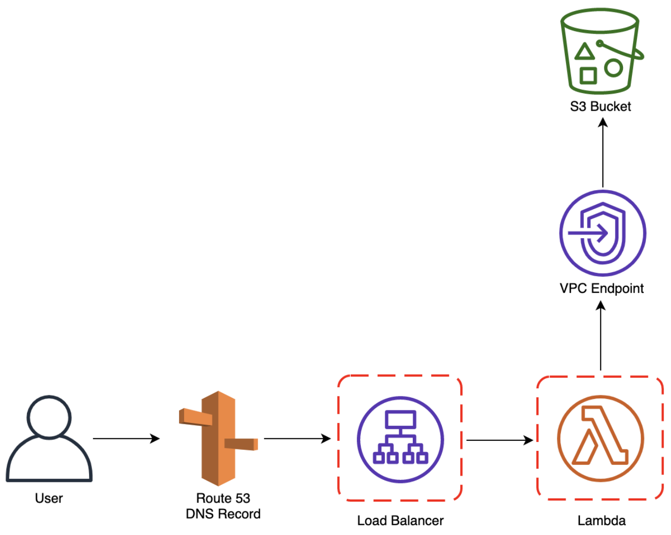
Even without some calculator at hand to predict costs of managed traffic, it is crystal clear that maintaining different CloudFront distributions for every feature and every frontend project can lead to useless complications from an AWS account's management perspective. It is worth mentioning that it's currently impossible to create two different origins from the main path and link them to two distinct domains within the same CloudFront distribution without incurring in a possible redirect that would lead to path problems mentioned above.

Solution

At first, we must notice that these deploys have an effimere nature. This means that a relatively short time of creation and removal for these infrastructural elements must be respected. To reach this goal it's useful to share the same resources between more than one interlocutors to minimize overhead.

The proposed solution is made thanks to the following infrastructural elements:

- 1 private S3 Bucket S3 (with Bucket policy)
- 1 LoadBalancer (with 2 Listener, 1 Target Group, 1 Security Group)
- 1 Lambda (in VPC with 1 Security Group)
- 1 S3 VPC Endpoint
- n DNS record in Route 53 (where n represents the number of features to validate at determined time)



Creation and configuration of the S3 Bucket

To obtain the desired result it’s imperative to create the S3 Bucket with the correct configuration. Following the definition of the CloudFormation template for this task:

```
S3Bucket:
 Type: AWS::S3::Bucket
 Properties:
   BucketName: 'subdomain.mydomain.com'
   CorsConfiguration:
     CorsRules:
       - AllowedHeaders:
           - '*'
         AllowedMethods:
           - GET
           - HEAD
           - POST
           - PUT
           - DELETE
         AllowedOrigins:
           - 'https://*.mydomain.com'
   PublicAccessBlockConfiguration:
     BlockPublicAcls: true
     BlockPublicPolicy: true
     IgnorePublicAcls: true
```

```
      RestrictPublicBuckets: true
    WebsiteConfiguration:
      ErrorDocument: error.html
      IndexDocument: index.html

S3BucketPolicy:
 Type: AWS::S3::BucketPolicy
 Properties:
   Bucket: !Ref S3Bucket
   PolicyDocument:
     Version: '2012-10-17'
     Statement:
       - Sid: VPCEndpointReadGetObject
         Effect: Allow
         Principal: "*"
         Action: s3:GetObject
         Resource: !Sub '${S3Bucket.Arn}/*'
         Condition:
           StringEquals:
             aws:sourceVpce: !Ref S3VPCEndpointId
```

It's worth noticing that "website configuration" has been enabled to allow HTTP requests towards the bucket but at the same time a Bucket Policy denies retrieving any kind of object from it unless a request is coming from the S3's VPC endpoint, ensuring that only allowed actors passing from the account's VPC can access that very bucket.

Creation and configuration of the Load Balancer

To allow final users to see the static website hosted on S3, a Load Balancer must be created (depending on the final user's nature, you can choose to keep the Load Balancer private or not). In CloudFormation this is the set of resources that must be created:

```
LoadBalancer:
 Type: AWS::ElasticLoadBalancingV2::LoadBalancer
 Properties:
   Name: !Sub '${ProjectName}'
   LoadBalancerAttributes:
     - Key: 'idle_timeout.timeout_seconds'
       Value: '60'
     - Key: 'routing.http2.enabled'
       Value: 'true'
     - Key: 'access_logs.s3.enabled'
       Value: 'true'
     - Key: 'access_logs.s3.prefix'
       Value: loadbalancers
     - Key: 'access_logs.s3.bucket'
       Value: !Ref S3LogsBucketName
   Scheme: internet-facing
   SecurityGroups:
     - !Ref LoadBalancerSecurityGroup
   Subnets:
     - !Ref SubnetPublicAId
     - !Ref SubnetPublicBId
```

```yaml
        - !Ref SubnetPublicCId
      Type: application

  LoadBalancerSecurityGroup:
   Type: AWS::EC2::SecurityGroup
   Properties:
      GroupName: !Sub '${ProjectName}-alb'
      GroupDescription: !Sub '${ProjectName} Load Balancer Security Group'
      SecurityGroupIngress:
        - CidrIp: 0.0.0.0/0
          Description: ALB Ingress rule from world
          FromPort: 80
          ToPort: 80
          IpProtocol: tcp
        - CidrIp: 0.0.0.0/0
          Description: ALB Ingress rule from world
          FromPort: 443
          ToPort: 443
          IpProtocol: tcp
      Tags:
        - Key: Name
          Value: !Sub '${ProjectName}-alb'
        - Key: Environment
          Value: !Ref Environment
      VpcId: !Ref VPCId

  HttpListener:
   Type: AWS::ElasticLoadBalancingV2::Listener
   Properties:
      DefaultActions:
        - RedirectConfig:
            Port: '443'
            Protocol: HTTPS
            StatusCode: 'HTTP_301'
          Type: redirect
      LoadBalancerArn: !Ref LoadBalancer
      Port: 80
      Protocol: HTTP

  HttpsListener:
   Type: AWS::ElasticLoadBalancingV2::Listener
   Properties:
      Certificates:
        - CertificateArn: !Ref LoadBalancerCertificateArn
      DefaultActions:
        - Type: forward
          TargetGroupArn: !Ref TargetGroup
      LoadBalancerArn: !Ref LoadBalancer
      Port: 443
      Protocol: HTTPS

  TargetGroup:
   Type: AWS::ElasticLoadBalancingV2::TargetGroup
   Properties:
      Name: !Sub '${ProjectName}'
      HealthCheckEnabled: false
      TargetType: lambda
      Targets:
        - Id: !GetAtt Lambda.Arn
   DependsOn: LambdaPermission
```

Thanks to this template, a public Load Balancer is deployed with a listener on port 80 (HTTP) redirecting on port 443 (HTTPS) with another listener which contacts a Target Group with a specific Lambda function registered on it.

Routing Lambda's code

Proposed solution's logic is managed by a Lambda Function. Below an example code can be found (being an example it's better to review and adapt code in case of production-ready solutions):

```
import json
from boto3 import client as boto3_client
from os import environ as os_environ
import base64
from urllib3 import PoolManager

http = PoolManager()
s3 = boto3_client('s3')

def handler(event, context):
   try:
       print(event)
       print(context)

       host = event['headers']['host']
       print("Host:", host)

       feature = host.split('.')[0]
       feature = "-".join(feature.split('-')[1:])
       print("Feature:", feature)

       path = event['path'] if event['path'] != "/" else "/index.html"
       print("Path:", path)

       query_string_parameters = event['queryStringParameters']
       query_string_parameters = [f"{key}={value}" for key, value in event['queryStri
ngParameters'].items()]
       print("Query String Parameters:", query_string_parameters)

       http_method = event["httpMethod"]
       url = f"http://{os_environ['S3_BUCKET']}.s3-website-eu-west-1.amazonaws.com/{f
eature}{path}{'?' if [] != query_string_parameters else ''}{'&'.join(query_string_para
meters)}"
       print(url)

       headers = event['headers']
       headers.pop("host")
       print("Headers:", headers)

       body = event['body']
       print("Body:", body)

       r = http.request(http_method, url, headers=headers, body=body)
       print("Response:", r)
```

```
        print("Response Data:", r.data)

        try:
            decoded_response = base64.b64encode(r.data).decode('utf-8')
        except:
            decoded_response =  base64.b64encode(r.data)

        print("Decoded Response:", decoded_response)
        print("Headers Response:", dict(r.headers))
        return {
            'statusCode': 200,
            'body': decoded_response,
            "headers": dict(r.headers),
            "isBase64Encoded": True
        }
    except Exception as e:
        print(e)
        return {
          'statusCode': 400
        }
```

Despite being a little tricky to read, operations done here are quite simple: starting from the DNS name which the user exploits to reach the Load Balancer, the Lambda Function processes the request to the S3 Bucket by building the feature’s appropriate subfolder to contact. To ensure the entire process works as expected, a DNS name for each feature must be created.

Conclusions

We showed how it is possible, thanks to a wise choice of configuration of a set of specific resources, to maintain a S3 Bucket private, despite having public webhosting enabled, with the goal of managing more versions of a static website: one for each defined sub-folder, reachable with a specific yet different DNS name. This approach is undoubtedly more fast and agile when the needs are to verify visible aspects of a site instead of configure its infrastructure avoiding time and management efforts of a CloudFront distribution.

Nice, isn’t it? 🙂 Write to us to deepen the topic!

beSharp

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all'avanguardia a realizzare progetti innovativi in campo IT.

Get in touch

beSharp.it
proud2becloud@besharp.it