AUTO-SCALING WEB APP

1. Create template EC2 instance with following attributes:
   1. Ubuntu AMI
   2. t2.micro instance type
   3. Create new SSH key pair (tc-websrv-jeremy)
   4. Create security group allowing SSH, HTTP, and HTTPS access
2. Set permissions on key file to 400
3. SSH into instance using key file
4. Install updates through apt
5. Install apache2
6. Use ufw to allow ‘Apache Full’ rules
7. Verify status of apche2 service in systemctl
8. Create simple HTML file in /var/www/html/test.html
9. Test successfully by loading http://[public IP of instance]/test.html
10. Disconnect SSH session
11. Right click instance in console and select Image and templates > Create image to generate custom AMI (jeremy-websrv-image)
12. Create autoscale group (jeremy-websrv-autoscale)
13. Create launch template (jeremy-websrv-launch-template) using custom AMI, t2.micro instance size, tc-websrv-jeremy key pair, and launch-wizard-20 security group
14. Set auto scaling group, application load balancer, and associated target group to all relate to launch template

GLOBAL LOAD BALANCING WITH AMAZON ROUTE 53

1. Register jeremytcawsproject.net domain in Route 53
2. Configure Route 53 health check (jeremy-health-check) to monitor http://jeremytcawsproject.net:80/test.html) and send alerts to my work email
3. Create A record with geolocation based routing pointing to dualstack.jeremy-alb-230417844.us-east-2.elb.amazonaws.com (jeremy-alb) and using health check
4. Successfully test web browser navigation to http://jeremytcawsproject.net/test.html
5. Configure CloudWatch alarm to monitor for any StatusCheckFailed within jeremy-websrv-autoscale and send notification to my work email when found

GLOBAL CONTENT DELIVERY WITH AMAZON CLOUDFRONT

1. Create S3 bucket (jeremy-static-site-bucket) with public access
2. Create and upload index.html and error.html
3. Enable static website hosting on bucket
4. Add an explicit bucket policy to allow read access to bucket objects for anyone
5. Create a CloudFront distribution (https://dlp5271gw2vro.cloudfront.net) using bucket as origin and using all edge locations for best global performance (unable to configure HTTPS because my user account lacks permission to generate TLS certs with ACM)
6. Create DynamoDB table (fortunes) and imported some data into it for dynamic site content
7. Create Lambda function (jeremy-serverless-function) with Python code to randomly select one of the values from the DynamoDB table to display to the user
8. Add permission to Lambda role for read only access to DynamoDB, then successfully run test of Lambda function
9. Upload new index.html, error.html, and cookie.jpg files for dynamic version of website, with index.html containing a call to Lambda function URL
10. Successfully test button on webpage triggers Lambda function to get a fortune value from the DynamoDB table