Load Testing and Auto-Scaling Project for High Availability of Web Server

This project demonstrates how to perform a load test using Postman and set up auto-scaling with AWS services, CloudWatch Alarms, a Cloudwatch custom dashboard for displaying custom metrics, and Lambda for reading metrics from the custom dashboard created in CloudWatch and sending reports via email (SES). Various Terraform modules are used to provision and manage AWS resources for a web application. This architecture ensures that your application can handle traffic fluctuations, and you receive email notifications when CPU utilization exceeds a specified threshold. It also uses a remote backend using AWS services like DynamoDB for storing Terraform locks and S3 for remote state files.

Objective

The primary objective of this project is to create a robust and scalable architecture for a web application that can handle traffic fluctuations and ensure high availability. It uses a combination of AWS services to achieve this goal.

The project leverages several AWS services to achieve its objectives:

VPC (Virtual Private Cloud): To create isolated network environments.

EC2 (Elastic Compute Cloud): To launch virtual servers.

ALB (Application Load Balancer): To distribute incoming traffic.

AutoScaling: To automatically adjust the number of EC2 instances based on traffic.

CloudWatch: To monitor and manage AWS resources.

S3 Bucket: To store data and assets.

DynamoDB: For storing Terraform locks.

Lambda: For serverless computing.

IAM (Identity and Access Management): For managing access to AWS resources.

SES (Simple Email Service): For sending email notifications.

Package Dependencies

Postman: Used for load testing.

PDFKIT, JINJA2 Python Libraries: Used for generating PDF documents.

AWS Infrastructure as Code with Terraform

The project's infrastructure is defined and managed as code using Terraform. It follows best practices for infrastructure as code and modularization. Here's an overview of the modules used:

1. VPC Configuration (VPC_New_Config)

Sets up a Virtual Private Cloud (VPC) with specified subnets, route tables, and other network configurations.

2. Security Groups (Security_Groups)

Creates security groups that define inbound and outbound traffic rules for resources, ensuring proper network security.

3. EC2 Instances (ec2_config)

Provisions EC2 instances within VPC subnets and attaches specified security groups, allowing for launching instances with desired configurations.

4. Launch Template (LaunchTemplate)

Defines instance specifications, such as instance type, AMI, and user data, for use in Auto Scaling groups.

5. Target Group (TargetGroup)

Creates an Elastic Load Balancer (ELB) target group, used for routing traffic to instances in the VPC.

6. Application Load Balancer (ALB)

Sets up an Application Load Balancer with specified subnets, security groups, and target group configuration.

7. Auto Scaling Group (autoscaling_group)

Configures Auto Scaling for instances, ensuring the desired number of instances are maintained based on load and instances launched from the specified launch template.

8. CloudWatch Alarm (CloudwatchAlarm)

Creates a CloudWatch Alarm that monitors CPU utilization of EC2 instances in the Auto Scaling group and creates an alarm when CPU utilization exceeds a specific threshold (e.g., CPU utilization > 80%).

9. Lambda Function (LambdaFunction)

Implements a Lambda function that reads the CloudWatch dashboard when an alarm is generated. It generates a PDF report with all relevant metrics and sends it via SES. The Lambda function is triggered by the CloudWatch Alarm.

Steps

1. Create a Load Test in Postman

- Open Postman and create a collection for your load test.
- Add a request to your collection targeting the URL of your Application Load Balancer (ALB).
- Create test scripts in Postman to simulate different load test scenarios, generating various requests with different payloads and parameters to simulate real-world traffic.

- Use Postman's collection runner to execute the load test by running the collection with multiple iterations and requests.

2. Set Up Auto Scaling with AWS

- Create an Auto Scaling group and configure it with a Launch Configuration or Launch Template.
- Set up scaling policies that trigger Auto Scaling based on CPU utilization exceeding a threshold (e.g., 90%). Configure scaling policies to add or remove instances from the Auto Scaling group.
- Create CloudWatch Alarms that monitor the CPU utilization metric and trigger when the threshold is exceeded.
- Configure the alarm to take action when it breaches its threshold, defining actions to trigger when the alarm state changes to "ALARM."

3. Create a Lambda Function

- Develop a Lambda function that sends email notifications. This function takes metrics from your custom CloudWatch dashboard and generates a PDF file with all metrics.
- The Lambda function should take parameters like the subject, recipient email address, email body, and include relevant logs, data from the CloudWatch custom dashboard, and timestamps.
- Set up the Lambda function to be triggered by a CloudWatch Alarm action. When the alarm state changes to "ALARM," the Lambda function is executed.
- In the Lambda function, use AWS Simple Email Service (SES) to send email notifications with the relevant data and logs.

4. Test and Verify

- Execute the load test in Postman to generate the desired traffic to your ALB.
- Monitor the CPU utilization in AWS CloudWatch to ensure it exceeds the defined threshold, triggering the Auto Scaling action.
- Verify that the CloudWatch Alarm changes to "ALARM" status, triggering the Lambda function.
- Check your email for the notification with logs, a PDF attachment, data, and timestamps.

Note

- Ensure that you have AWS credentials and Terraform installed. You can configure your AWS credentials using AWS CLI or environment variables.
- Review and adjust the variables and configurations in the respective module source directories to match your specific requirements.

Usage

Terraform Configuration:

Configure the Terraform modules and variables in your project according to your application's specific requirements.

Terraform Init:
Run terraform init to initialize the working directory.

Terraform Plan:
Use terraform plan to review the execution plan and verify that the infrastructure changes align with your expectations.

Terraform Apply:
Execute terraform apply to create the AWS resources based on the configuration defined in the Terraform modules.

Terraform Destroy:
If necessary, you can use terraform destroy to tear down the infrastructure.

Monitoring and Maintenance

- Continuously monitor and maintain the resources for optimal performance, security, and availability. Configure CloudWatch alarms and monitoring for key metrics