Deploy a Scalable and Highly Available Secured Webapp on AWS Cloud.

TABLE OF CONTENTS:

Project

The goal of your project is to deploy a scalable, highly available, and secured web application in a 3-tier architecture and provide access to the application for end users from the public internet. Let's break down this goal into more detail:

1.Scalability: Scalability refers to the ability of your web application to handle increased traffic and user demand without compromising performance. By implementing a scalable architecture, you can ensure that your application can automatically adjust its resources to accommodate varying loads.

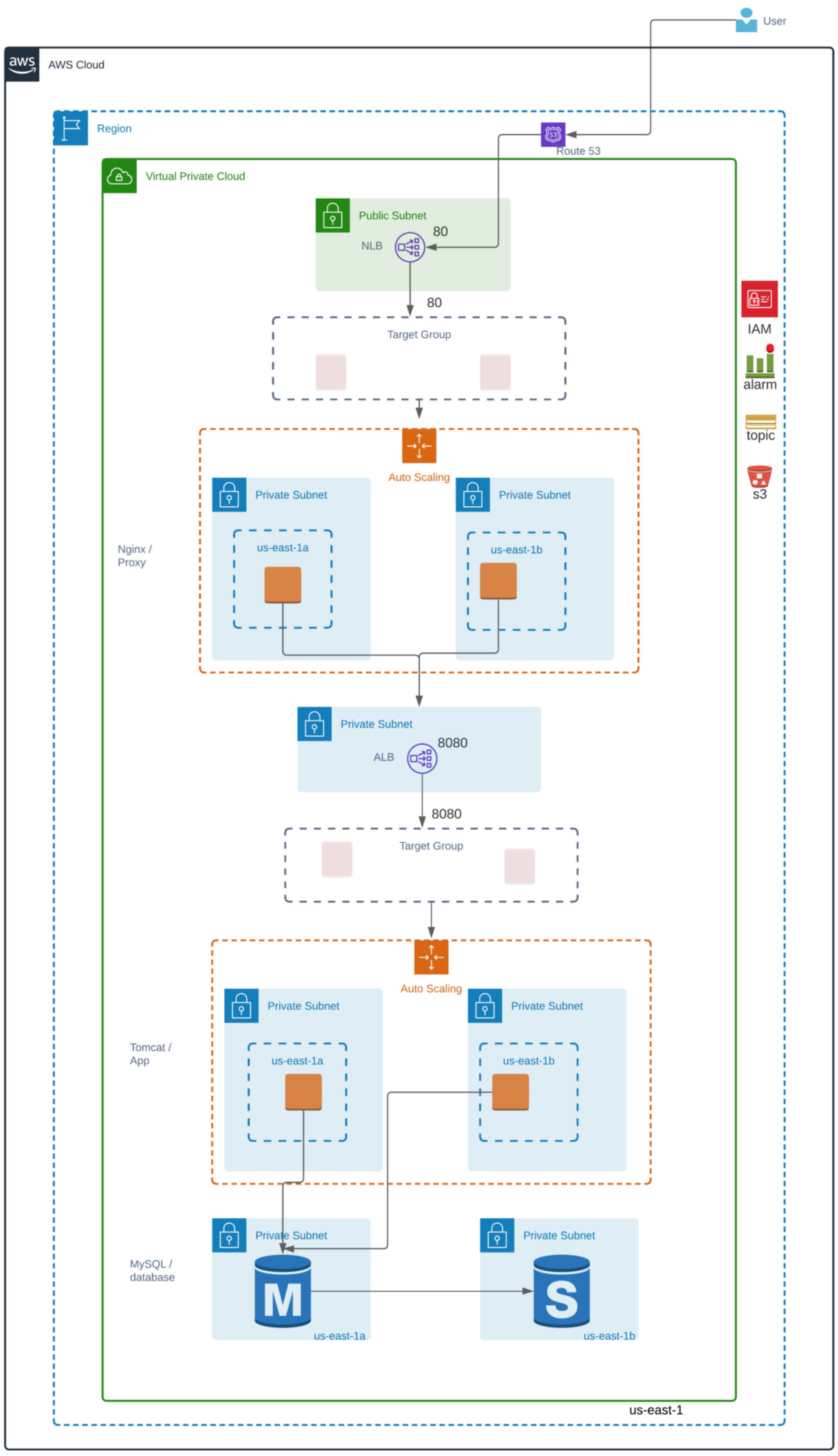
2.High Availability: High availability ensures that your web application remains accessible and functional even in the event of failures or disruptions. This involves deploying redundant components and implementing mechanisms for automatic failover and load balancing.

3.Secured Web Application: Security is a crucial aspect of any web application. It involves implementing measures to protect your application and its data from unauthorized access, vulnerabilities, and attacks. This can include implementing security groups, encryption, access controls, and other security best practices.

4. 3-Tier Architecture: The 3-tier architecture separates your web application into three layers: presentation, application, and data layers. This modular approach allows for better organization, scalability, and maintainability of your application. The presentation layer handles user interface and interaction, the application layer manages the application's logic and functionality, and the data layer stores and retrieves data.

5.End User Access from Public Internet: Once your web application is deployed, it should be accessible to end users from the public internet. This means that users can access and interact with your application using standard web browsers or other client applications.

By achieving these goals, you will have a web application that can handle increased traffic, remain available even during failures, ensure the security of user data, follow a modular architecture for better maintenance, and provide access to end users from the public internet. This will enable a smooth and reliable user experience while ensuring the application's performance and integrity.


User
AWS Cloud
Region
Route 53
Virtual Private Cloud
Public Subnet
NLB
80
80
Target Group
Auto Scaling
Private Subnet
Private Subnet
us-east-1a
us-east-1b
Nginx /
Proxy
Private Subnet
ALB
8080
8080
Target Group
Auto Scaling
Private Subnet
Private Subnet
us-east-1a
us-east-1b
Tomcat /
App
Private Subnet
Private Subnet
MySQL /
database
M
S
us-east-1a
us-east-1b
us-east-1
IAM
alarm
topic
s3

Resources:

Below are the resources required to deploy this project.

Here's an explanation of how each of the mentioned resources can be used in your project:

1. VPC (Virtual Private Cloud): VPC is a virtual network dedicated to your AWS account. It provides isolated and secure network environments for your resources. In your project, you would create a VPC to house your application components and control network settings such as IP ranges and subnets.
2. Subnets: Subnets are subdivisions of a VPC and allow you to partition your network. You can create public and private subnets to segregate resources based on their accessibility. Public subnets are used for components that require direct internet access, while private subnets are used for backend resources that should not be exposed to the internet directly.
3. Route Tables: Route tables determine how network traffic is directed within a VPC. You would configure route tables to specify the destinations for traffic and how it should be routed, such as routing internet-bound traffic to an internet gateway.
4. Internet Gateway: An internet gateway is used to enable internet connectivity for resources within your VPC. It acts as a gateway between your VPC and the internet, allowing communication to and from the public internet.
5. NAT Gateway: A NAT (Network Address Translation) gateway allows resources in private subnets to access the internet while remaining hidden from inbound connections. It provides a secure way for resources in private subnets to communicate with the internet.
6. Security Groups: Security groups act as virtual firewalls for your EC2 instances and control inbound and outbound traffic. You would configure security groups to allow or restrict access to your resources based on protocols, ports, and source IP addresses.
7. Network ACL: Network ACLs are another layer of network security in AWS. They act as stateless firewalls at the subnet level, controlling inbound and outbound traffic based on rules that you define.
8. Elastic IP: An Elastic IP is a static, public IPv4 address that you can associate with your instances. It allows your application to have a fixed IP address even if the instance is stopped and restarted.
9. S3 Bucket: Amazon S3 (Simple Storage Service) is a scalable object storage service. You can use an S3 bucket to store static assets, media files, backups, or any other data your application requires.
10. SNS Topic: Amazon SNS (Simple Notification Service) enables you to send notifications to various subscribers, such as email, SMS, or HTTP endpoints. You could use an SNS topic to send notifications based on specific events or triggers in your application.

11. CloudWatch Alarms: Amazon CloudWatch provides monitoring and management for your AWS resources. You can create CloudWatch alarms to monitor specific metrics and trigger actions based on thresholds or anomalies. In your project, you can configure alarms to scale your resources based on CPU utilization or other metrics.

12.EC2 Instances: Amazon EC2 (Elastic Compute Cloud) provides virtual server instances on AWS. You would deploy EC2 instances to host your web application, both for the front-end (Nginx) and the application logic (Tomcat).

13.RDS: Amazon RDS (Relational Database Service) is a managed database service. You can use RDS to deploy and manage a relational database (such as MySQL) for your application's data storage.

14.Auto Scaling Group: Auto Scaling allows you to automatically adjust the number of EC2 instances based on demand. You would configure an Auto Scaling group to ensure that your application can scale up or down based on CPU utilization or other metrics.

15.Load Balancer: Load balancers distribute incoming traffic across multiple EC2 instances, enhancing the availability and scalability of your application. You would deploy a load balancer in front of Nginx and Tomcat to handle traffic distribution.
16.Route53 Hosted Zone: Amazon Route 53 is a scalable domain name system (DNS) web service. You would create a Route53 hosted zone to manage the DNS records for your domain name and route incoming traffic to the appropriate resources, such as the load balancer.

17.IAM Policy: AWS Identity and Access Management (IAM) policies define permissions and access controls for users, groups, or roles in your AWS account. You would create an IAM policy to grant specific permissions required for your resources, such as accessing S3 buckets, EC2 instances, or RDS databases.

18.IAM Role: IAM roles are entities that you can create in IAM to enable trusted access to AWS resources. Roles are used to delegate permissions to AWS services, applications, or other accounts. In your project, you may create an IAM role to allow EC2 instances to access other AWS services, such as S3 or RDS.

19.Session Manager: AWS Systems Manager Session Manager allows you to securely manage your instances without the need for SSH or RDP access. You can use Session Manager to establish a secure shell (SSH) connection to your EC2 instances for administrative tasks.

20.Java Application: The Java application represents the code and logic of your web application. You would package your application as a Java Archive (JAR) file or a web application archive (WAR) file and deploy it on Tomcat instances.

These resources collectively provide the infrastructure and services needed to deploy and operate a scalable, highly available, and secured web application. They enable network configuration, security, monitoring, database management, load balancing, DNS management, access control, and application hosting.

Skills:

Below are the skills required to complete this project deployment to deploy infrastructure and application components.

1. AWS
2. Nginx
3. Apache Tomcat
4. Apache Maven
5. MySQL

1.AWS: You would use AWS services to deploy and manage the infrastructure components of your web application. For example:

EC2 instances: Deploy and manage virtual server instances to host your Nginx and Tomcat servers.

VPC: Create a virtual network environment to isolate and secure your resources.

RDS: Set up a MySQL database instance to store and manage your application data.

S3: Store static assets, media files, or backups.

Auto Scaling: Automatically adjust the number of EC2 instances based on traffic demands.

Route53: Manage your domain's DNS records and route incoming traffic to your application.

Nginx: Nginx would serve as the front-end server in your architecture. Its role includes:

Receiving incoming HTTP/HTTPS requests from end users.

Load balancing and distributing traffic across multiple backend servers.

Handling SSL/TLS termination for secure connections.

Caching static content to improve performance.

Acting as a reverse proxy for routing requests to the appropriate backend servers (e.g., Tomcat).

3.Apache Tomcat: Tomcat is the Java application server responsible for hosting your Java web application. Its functionalities include:

Running Java servlets and Java Server Pages (JSP) that form the core of your application.

Managing the execution environment for your Java application.

Handling HTTP requests and generating responses.

Providing features like session management and connection pooling for database access.

4.Apache Maven: Maven is a build automation tool that simplifies the management of dependencies and the build process for your Java application. It helps by:

Defining and managing project dependencies, such as libraries and frameworks used in your application.

Compiling your source code and generating executable artifacts.

Automating the packaging of your application into a deployable format, such as a JAR or WAR file.

Assisting in running tests and generating reports.

5.MySQL: MySQL is used as the relational database management system for your application. Its role includes:

Storing and managing your application's data, such as user information, content, and settings.

Providing a structured way to organize and query your data using SQL.

Ensuring data integrity and enforcing relationships between different entities.

Supporting concurrent access to the database, allowing multiple users to interact with your application simultaneously.

By leveraging these technologies, you can deploy a scalable and secure web application architecture. AWS provides the necessary infrastructure, Nginx handles the front-end traffic and load balancing, Tomcat hosts the Java application, Maven simplifies the build process, and MySQL manages the data storage and retrieval.

Configuration

All AWS resources required for this project deployment are in the scope of AWS Free Tier, except Multi-AZ RDS, NAT and Route 53 Domain.

Deployment

Below are the high-level steps to implement the project.
step-by-step breakdown of the deployment process for your scalable, highly available, and secured web application in AWS, based on the 3-tier Architecture.

1.Create Golden AMI for front and app tier:
Launch EC2 instances for the front-end and application tiers.
Configure the instances with the necessary software, including Nginx for the front-end tier and Tomcat for the application tier.
Customize the configuration and settings according to your requirements.
Create an Amazon Machine Image (AMI) of each instance to serve as the base image for future deployments.

2.Deploy a public load balancer in front of Nginx:
Create an Elastic Load Balancer (ELB) or use the Application Load Balancer (ALB) service.
Configure the load balancer to distribute incoming traffic to the Nginx instances in the front-end tier.
Enable SSL termination, if required, for secure HTTPS connections.

3.Deploy an auto scaling group for Nginx:
Create an Auto Scaling group for the front-end tier instances running Nginx.
Configure the Auto Scaling group to automatically scale the number of instances based on traffic demands.
Set up scaling policies based on CPU utilization to add or remove instances dynamically.

4.Deploy a private load balancer in front of Tomcat:
Create a Network Load Balancer (NLB) or use the Application Load Balancer (ALB) service.

Configure the load balancer to distribute traffic to the Tomcat instances in the application tier.
Enable SSL termination, if required, for secure communication between the load balancer and Tomcat instances.

5.Deploy an auto scaling group for Tomcat:
Create an Auto Scaling group for the application tier instances running Tomcat.
Configure the Auto Scaling group to automatically adjust the number of instances based on traffic demands.
Set up scaling policies based on CPU utilization to scale the instances in or out as needed.

6.Deploy RDS into private subnets:
Create an RDS instance in the desired Availability Zone (AZ) and configure it to use MySQL as the database engine.
Place the RDS instance in the private subnets to enhance security and isolate it from the public internet.
Set up the appropriate security groups and access control to allow access from the application tier instances.

7.Configure SNS notification on Auto Scaling Group event change:
Set up an SNS (Simple Notification Service) topic to receive notifications from the Auto Scaling group.
Configure the Auto Scaling group to send notifications to the SNS topic when scaling events occur.
Subscribe to the SNS topic to receive notifications via email, SMS, or other preferred communication methods.

8.Configure scaling policies based on CPU utilization:
Define scaling policies for both the front-end and application tiers.
Set the scaling thresholds, such as 80% CPU utilization, to trigger scaling actions.
Specify the scaling actions, such as adding or removing instances, based on the defined thresholds.

9.Customize Launch Configuration user-data:
Customize the user-data section of the Launch Configuration for both the Nginx and Tomcat instances.
Use user-data scripts to automate the provisioning and configuration of the software, such as pulling artifacts from J Frog, S3, or other sources.

Configure the scripts to install and set up Nginx and Tomcat, including any required dependencies or configurations.

By following these steps, you will be able to deploy a scalable, highly available, and secured web application in AWS cloud, utilizing the 3-tier architecture and the mentioned services and technologies.

Verification:

Here are the details for verifying the administrator login to EC2 instances through Session Manager and testing end-user access to the web services from a public internet browser:

1. Verify administrator login to EC2 instances from Session Manager:
    - Ensure that the IAM user or role you are using has the necessary permissions to access EC2 instances through AWS Systems Manager Session Manager.
    - Open the AWS Management Console and navigate to the EC2 service.
    - Select the EC2 instance you want to access and choose the "Connect" button.
    - In the dropdown menu, select "Session Manager" as the connection method.
    - Click on the "Connect" button to establish a secure shell (SSH) session to the EC2 instance using Session Manager.
    - Once connected, you should be able to execute commands and perform administrative tasks on the EC2 instance.
2. Verify end-user access to web services from a public internet browser:
    - Ensure that the public load balancer in front of Nginx is properly configured and that the DNS records for your domain are set up correctly.
    - Open a web browser and enter the URL or domain name associated with your web application.
    - The public load balancer will receive the request and distribute it to the available Nginx instances in the front-end tier.
    - The Nginx instances will handle the request and forward it to the appropriate backend components, such as Tomcat.
    - The web page or application should load successfully in the browser, indicating that end users can access the web services from the public internet.

By performing these verifications, you can ensure that as an administrator, you can securely access the EC2 instances using Session Manager, and as an end user, you can access the web services of your application through a public internet browser. It's important to validate these steps to ensure that the deployment is functioning correctly and that users can access your web application as intended.