**65)Availability,Scalability,Elasticity**

1. **Scalability**:
   * **Definition**: The ability of a system to handle increased loads by either making existing resources stronger (vertical scaling) or adding more resources (horizontal scaling).
   * **Vertical Scaling**: Upgrading an existing server to a more powerful one (e.g., moving from a small EC2 instance to a larger one).
   * **Horizontal Scaling**: Adding more servers to handle more requests (e.g., adding more EC2 instances to distribute the load).
2. **High Availability**:
   * **Definition**: Ensuring that your application remains operational even if one part of it fails. This is achieved by running your application across multiple Availability Zones (AZs) in AWS.
   * **Example**: If you have a call center in New York and another in San Francisco, if the New York center goes down, the San Francisco center can still take calls.
3. **Elasticity**:
   * **Definition**: A cloud-native feature that allows a system to automatically scale up or down based on current demand. This means you only use (and pay for) the resources you need at any given time.
   * **Example**: If your website experiences a sudden spike in traffic, AWS can automatically add more EC2 instances to handle the load and then reduce them when traffic decreases.
4. **Agility**:
   * **Definition**: The ability to quickly provision IT resources, allowing organizations to respond faster to changing business needs. This means developers can get the resources they need in minutes instead of weeks.
   * **Example**: With AWS, a developer can launch a new server with just a few clicks, enabling rapid development and deployment of applications.

**Summary of How They Work Together**

* **Elastic Load Balancing**: Distributes incoming traffic across multiple EC2 instances to ensure no single instance is overwhelmed, enhancing both scalability and high availability.
* **Auto Scaling Groups**: Automatically adjusts the number of EC2 instances based on current demand, ensuring elasticity. If traffic increases, it adds instances; if traffic decreases, it removes them.
* **High Availability**: By running instances in multiple AZs, your application can withstand failures in one location, ensuring continuous operation.

**Example Scenario**

Imagine you run an online store:

* **Scalability**: During a sale, you notice a spike in traffic. You can either upgrade your existing server (vertical scaling) or add more servers (horizontal scaling) to handle the increased load.
* **High Availability**: You have servers in both New York and San Francisco. If the New York server goes down, customers can still shop through the San Francisco server.
* **Elasticity**: Your online store automatically adds more servers during peak shopping hours and reduces them during off-peak hours, so you only pay for what you use.
* **Agility**: When you want to launch a new feature, you can quickly provision the necessary resources in AWS, allowing your team to innovate faster.

**66)Elastic Load balancer:-**

**Elastic Load Balancing (ELB)**

**Definition**: Elastic Load Balancing is a managed service by AWS that automatically distributes incoming application traffic across multiple targets, such as EC2 instances, containers, and IP addresses. This helps ensure that no single instance is overwhelmed with too much traffic, improving the availability and fault tolerance of your application.

**Key Features of ELB:**

1. **Traffic Distribution**: ELB routes incoming traffic to multiple backend instances, ensuring that the load is balanced. This helps in maintaining performance and availability.
2. **Health Checks**: ELB performs regular health checks on the registered instances. If an instance fails the health check, the load balancer stops sending traffic to it until it becomes healthy again.
3. to healthy instances

**67)ALB HANDS ONNN:-**

**Step 1: Launch EC2 Instances**

1. **Log in to the AWS Management Console**.
2. **Navigate to EC2**: Click on "EC2" under the "Services" menu.
3. **Launch Instances**:
   * Click on "Launch Instance."
   * Choose an Amazon Machine Image (AMI) (e.g., Amazon Linux 2).
   * Select an instance type (e.g., t2.micro).
   * Configure instance details (e.g., number of instances, network settings).
   * Add storage (default settings are usually fine).
   * Configure security group settings to allow HTTP (port 80) and SSH (port 22) access.
   * Review and launch the instances.
   * **User Data**: If you want to run a script on startup (e.g., to install a web server), you can add it in the "User Data" section.

**Step 2: Verify EC2 Instances**

1. **Check Instance Status**: Once the instances are launched, go to the "Instances" section and ensure that both instances are in the "running" state.
2. **Get Public IP Addresses**: Copy the public IPv4 addresses of both instances.

**Step 3: Test EC2 Instances**

1. **Access Instances**: Open a web browser and enter the public IP address of each instance. You should see a "Hello World" message if a web server is running.

**Step 4: Create a Load Balancer**

1. **Navigate to Load Balancers**: In the EC2 dashboard, click on "Load Balancers" in the left sidebar.
2. **Create Load Balancer**:
   * Click on "Create Load Balancer."
   * Choose **Application Load Balancer (ALB)** for HTTP/HTTPS traffic.
   * **Name**: Give your load balancer a name (e.g., DemoALB).
   * **Scheme**: Select "Internet-facing" to allow public access.
   * **IP Address Type**: Choose IPv4.
   * **Availability Zones**: Select the availability zones where your EC2 instances are running. Ensure that you check all the AZs for high availability.

**Step 5: Configure Security Group for Load Balancer**

1. **Create Security Group**:
   * Create a new security group (e.g., demo-sg-load-balancer).
   * Add an inbound rule to allow HTTP traffic (port 80) from anywhere (0.0.0.0/0).
   * Outbound rules can remain as default (allow all traffic).
2. **Assign Security Group**: Select the newly created security group for your load balancer.

**Step 6: Configure Listeners and Target Groups**

1. **Listeners**:
   * Set up a listener for HTTP on port 80.
   * Choose to forward traffic to a target group.
2. **Create Target Group**:
   * Click on "Create a target group."
   * Name the target group (e.g., demo-tg-alb).
   * Set the protocol to HTTP and the port to 80.
   * Configure health checks (default settings are usually fine).
3. **Register Targets**:
   * Register your EC2 instances in the target group by selecting them and clicking "Add to registered."
   * Click "Create" to finalize the target group.

**Step 7: Complete Load Balancer Setup**

1. **Review and Create**: Review your load balancer settings and click "Create."
2. **Wait for Provisioning**: The load balancer will take a few moments to provision. Once active, it will provide a DNS name.

**Step 8: Test Load Balancer**

1. **Access Load Balancer**: Copy the DNS name of the load balancer and paste it into a web browser.
2. **Verify Load Balancing**: Refresh the page multiple times. You should see responses alternating between the two EC2 instances, indicating that the load balancer is distributing traffic.

**Step 9: Monitor Target Health**

1. **Check Target Group**: Go to the target group you created and check the health status of the registered instances. Both should be marked as healthy.
2. **Simulate Instance Failure**: Stop one of the EC2 instances and observe the target group. The stopped instance should show as unhealthy, and the load balancer will stop sending traffic to it.

**Step 10: Restart Instance**

1. **Start the Stopped Instance**: Go back to the EC2

**You**

CONTINUW

**BLACKBOXAI**

Certainly! Let's continue from where we left off in the step-by-step process of setting up Elastic Load Balancing with EC2 instances.

**Step 10: Restart Instance (Continued)**

1. **Start the Stopped Instance**:
   * Go back to the EC2 dashboard.
   * Select the instance that you previously stopped.
   * Click on the "Instance State" dropdown and select "Start Instance."
   * Wait for the instance to transition back to the "running" state.
2. **Verify Health Status**:
   * Return to the target group in the Load Balancers section.
   * Refresh the page to check the health status of the targets.
   * The previously stopped instance should now show as healthy again.

**Step 11: Final Testing**

1. **Test Load Balancer Again**:
   * Copy the DNS name of the load balancer again and paste it into a web browser.
   * Refresh the page multiple times to ensure that both EC2 instances are responding correctly.
   * You should see the "Hello World" message from both instances, confirming that the load balancer is functioning properly.

**Step 12: Clean Up Resources**

1. **Terminate EC2 Instances**:
   * If you no longer need the instances, go to the EC2 dashboard.
   * Select both instances, click on "Instance State," and choose "Terminate."
   * Confirm the termination to avoid incurring charges.
2. **Delete Load Balancer**:
   * Navigate back to the Load Balancers section.
   * Select the load balancer you created (DemoALB).
   * Click on "Actions" and select "Delete."
   * Confirm the deletion to clean up resources.
3. **Delete Target Group**:
   * Go to the Target Groups section.
   * Select the target group (demo-tg-alb) you created.
   * Click on "Actions" and select "Delete."
   * Confirm the deletion.
4. **Delete Security Group**:
   * If you created a new security group specifically for the load balancer, you can delete it as well.
   * Go to the Security Groups section in the EC2 dashboard.
   * Select the security group (demo-sg-load-balancer) and click on "Actions" > "Delete."
   * Confirm the deletion.

**Summary of Key Concepts**

* **Elastic Load Balancing (ELB)**: A service that automatically distributes incoming application traffic across multiple targets (EC2 instances) to ensure high availability and fault tolerance.
* **Application Load Balancer (ALB)**: Designed for HTTP/HTTPS traffic, it operates at Layer 7 and provides advanced routing features.
* **Target Groups**: A way to group EC2 instances that the load balancer routes traffic to. Health checks ensure that only healthy instances receive traffic.
* **Auto Scaling**: While not covered in this specific setup, integrating Auto Scaling Groups with your load balancer allows for dynamic scaling of EC2 instances based on demand.

**Conclusion**

By following these steps, you have successfully set up an Elastic Load Balancer with multiple EC2 instances, allowing for efficient traffic distribution and high availability. This setup is crucial for modern applications that require scalability and resilience in the cloud.

**68)Auto Scaling Group:-**

**Key Concepts**:

1. **Auto Scaling Group (ASG)**:
   * **Definition**: A service that automatically adjusts the number of EC2 instances in response to changing demand.
   * **Scaling Out**: Adding more EC2 instances when demand increases (e.g., during peak shopping hours).
   * **Scaling In**: Removing EC2 instances when demand decreases (e.g., during off-peak hours).
   * **Minimum and Maximum Size**: You can set a minimum number of instances to always run and a maximum limit to control costs.
2. **Integration with Load Balancer**:
   * **Load Balancer Role**: Distributes incoming traffic across all healthy EC2 instances in the ASG.
   * **Health Checks**: The ASG monitors the health of instances. If an instance becomes unhealthy, it is terminated and replaced with a new instance.
   * **Cost Efficiency**: By scaling in and out based on demand, ASGs help optimize costs, ensuring that you only pay for the resources you need.
3. **Benefits**:
   * **Elasticity**: Automatically adjusts capacity to match demand, ensuring optimal performance.
   * **High Availability**: By distributing traffic and replacing unhealthy instances, ASGs contribute to the overall availability of applications.

**Sample Interview Questions**:

* How does an Auto Scaling Group work in AWS?
* What are the benefits of using Auto Scaling Groups with Elastic Load Balancing?
* Can you explain the difference between scaling out and scaling in?

**Hands-On Point of View**

**Step-by-Step Process**:

1. **Create EC2 Instances**:
   * Launch a few EC2 instances that will serve as the backend for your application.
2. **Set Up a Load Balancer**:
   * Create an Application Load Balancer (ALB) to distribute traffic among the EC2 instances.
   * Configure the load balancer to route traffic to the target group containing your EC2 instances.
3. **Create an Auto Scaling Group**:
   * Define the ASG with a minimum, desired, and maximum number of instances.
   * Choose the launch configuration or launch template that specifies the instance type, AMI, and other settings.
4. **Configure Scaling Policies**:
   * Set up scaling policies based on CloudWatch metrics (e.g., CPU utilization). For example, scale out when CPU usage exceeds 70% and scale in when it drops below 30%.
5. **Test the Setup**:
   * Simulate increased traffic to see how the ASG scales out by adding more instances.
   * Monitor the load balancer to ensure it distributes traffic evenly across all instances.
   * Stop one of the instances to test the ASG's ability to replace unhealthy instances.
6. **Monitor and Optimize**:
   * Use AWS CloudWatch to monitor the performance of your ASG and load balancer.
   * Adjust scaling policies and instance types based on observed performance and cost.

**Hands-On Example**:

* **Scenario**: You have an e-commerce application that experiences high traffic during sales events.
* **Implementation**: Set up an ASG with a minimum of 2 instances and a maximum of 10. Configure the load balancer to distribute traffic. As traffic increases, the ASG automatically adds instances, ensuring that the application remains responsive.

**69)ASG HandsOn:-**

This detailed walkthrough provides step-by-step instructions on how to set up an **Auto Scaling Group (ASG)** in AWS, demonstrating various features and options available. I'll break down every line of the process for you.

**1. Terminating Existing Instances**

* **Action**: You start by terminating the first two EC2 instances that were previously set up. This is necessary because we are going to set up an Auto Scaling Group, and it will manage instances for you.
* **Reason**: Auto Scaling Groups are designed to automatically create and manage EC2 instances based on scaling policies, so the current instances are no longer needed.

**2. Creating an Auto Scaling Group**

* **Action**: Navigate to **Auto Scaling Groups** in the AWS console to create a new ASG.
* **Step**: Select **Create Auto Scaling Group** and name it DemoASG.
* **Reason**: You need to give the ASG a name, so AWS can identify and manage it.

**3. Creating a Launch Template**

* **Action**: The ASG needs a **Launch Template** to define the configuration for the EC2 instances it will create. You name it DemoLaunchTemplate.
* **Step**: The launch template specifies the instance configuration like OS, instance type, security group, etc.
* **Reason**: Without a launch template, the ASG would not know how to create new EC2 instances.

**4. Configuring Launch Template Details**

* **Amazon Machine Image (AMI)**: Choose **Amazon Linux 2** as the base AMI for the EC2 instances.
* **Instance Type**: Set the instance type as t2.micro, which is a small instance type suitable for testing.
* **Key Pair**: You decide not to use a key pair for SSH access (common for web servers where access is done via a load balancer).
* **Security Group**: Select an existing security group (e.g., launch-wizard-1), which controls the inbound and outbound traffic to the EC2 instances.
* **EBS Volumes**: Leave storage settings as default since no custom volumes are required.
* **User Data**: You add **user data** to the EC2 instances, which can run scripts or commands automatically when the instance starts. This is where you can set up initialization tasks like installing software.

**5. Creating the Launch Template**

* **Action**: After defining the template details, you create the launch template. This serves as the configuration blueprint for launching EC2 instances.

**6. Choosing Where to Launch Instances**

* **Action**: In the ASG setup, you select the **VPC** where the instances will reside. You choose multiple availability zones (AZs) to ensure high availability.
* **Reason**: Using multiple AZs allows you to distribute instances across different physical locations within AWS to ensure redundancy.

**7. Configuring Load Balancers**

* **Action**: You choose to attach the Auto Scaling Group to an existing **Application Load Balancer (ALB)**. This ensures that when EC2 instances are created, they are automatically registered with the target group for the load balancer.
* **Health Checks**: You set the health checks to monitor both EC2 instance health and the health of the instances in the ALB target group. This allows you to ensure that traffic is only sent to healthy instances.

**8. Setting Scaling Parameters**

* **Desired Capacity**: You set the **desired capacity** of the Auto Scaling Group to 2 instances. This is the ideal number of instances you want running at all times.
* **Minimum Capacity**: The **minimum capacity** is set to 1, meaning if the Auto Scaling Group scales down, it will never have fewer than one instance.
* **Maximum Capacity**: The **maximum capacity** is set to 4, so the ASG will never scale up beyond 4 instances.
* **Scaling Policies**: Although not covered in this step, you can define scaling policies later to automatically adjust the number of instances based on metrics like CPU utilization or incoming traffic.

**9. Reviewing and Creating the Auto Scaling Group**

* **Action**: You review all the settings you’ve configured and then create the Auto Scaling Group.
* **Reason**: This finalizes the configuration and triggers the creation of the Auto Scaling Group with the desired parameters.

**10. Viewing Activity in the ASG**

* **Action**: After creating the ASG, you observe the activity history, where the system indicates that EC2 instances are being launched to meet the desired capacity of 2 instances.
* **Reason**: The Auto Scaling Group automatically starts creating the number of instances needed to satisfy the desired capacity. It also monitors the state of the instances in real time.

**11. Observing EC2 Instances**

* **Action**: Go to the **EC2 Dashboard** and see that two new EC2 instances are created by the Auto Scaling Group. These instances are now fully managed by the ASG.
* **Reason**: The Auto Scaling Group ensures that the number of instances remains at the desired level.

**12. Checking Target Group Registration**

* **Action**: You verify that the instances created by the ASG have been automatically registered with the target group of your load balancer. You see that both instances are listed under the target group.
* **Reason**: Integration between the ASG and ALB ensures that new instances are automatically included in the load balancer’s target group, enabling traffic routing.

**13. Health Check Configuration**

* **Action**: You adjust the **health check** settings to make the process faster by setting the healthy threshold to 2 and reducing the interval to 5 seconds.
* **Reason**: This speeds up the process of detecting and marking instances as healthy after launch. It’s important to have quick health checks for faster traffic routing.

**14. Observing the Health of Instances**

* **Action**: After adjusting the health check, you observe that both instances are now marked as healthy.
* **Reason**: Once instances pass the health check, they become eligible to receive traffic from the load balancer.

**15. Testing the Load Balancer**

* **Action**: You test the setup by visiting the load balancer's DNS name and verifying that it returns a "Hello World" message from both EC2 instances.
* **Reason**: This confirms that both instances are healthy and serving traffic via the load balancer.

**16. Terminating an Instance**

* **Action**: You manually terminate one of the instances to simulate a failure scenario.
* **Reason**: You want to see how the Auto Scaling Group responds to instance termination (e.g., scaling out to replace the terminated instance).

**17. Auto Scaling Group Reacts**

* **Action**: The ASG detects that an instance is no longer in service and immediately starts a new instance to replace the terminated one. The activity history shows the launch of a new instance.
* **Reason**: The ASG ensures that the desired capacity is maintained, automatically replacing terminated or unhealthy instances.

**18. Viewing New EC2 Instance**

* **Action**: The new instance is launched and moves into the "pending" state, while the terminated instance is marked as "terminated" in the EC2 dashboard.
* **Reason**: The Auto Scaling Group ensures that the number of instances remains at the desired capacity, replacing instances automatically as needed.

**19. Understanding Auto Scaling Group Behavior**

* **Action**: The overall concept behind Auto Scaling Groups is demonstrated — automatically scaling in and out based on demand, ensuring the desired number of healthy instances is always running.
* **Reason**: This is the main advantage of using Auto Scaling Groups — they allow for automatic scaling and recovery from instance failures.

**20. Exploring Scaling Policies (Optional)**

* **Action**: Though not demonstrated here, scaling policies can be set up later to automatically adjust the desired capacity based on metrics like CPU usage or request count.
* **Reason**: This gives the Auto Scaling Group the ability to dynamically adjust to varying loads without manual intervention.

**Conclusion:**

This step-by-step guide walks through the process of creating an Auto Scaling Group in AWS, from setting up the launch template to configuring scaling parameters and testing how the ASG manages EC2 instances. The core advantage of ASGs is their ability to automatically scale the number of instances up or down based on demand, ensuring high availability and efficient resource management.

**70)Auto scaling groups -Scaling Strategies:**

**Manual Scaling:-** we should change according to demand

Dynamic Scaling:- Respond to changing Demand that is manually

Simple/step Scaling:if cloud watch alarm triggered (Example CPU>70%) then add 2 Units When Cloud watch Alarm Triggered(Example CPU<30%) then remove

Target Tracking:-

I want average of 40 % that means when traffic is sent the traffic will be in between that 40 %

Scheduled Scaling:Scaling on schedule that means on festival time

Predictive Scaling:Scaling based on Past patterns .Automatically provisions right number of EC2 instances