AWS Firewalls & ASG

AWS firewalls:

Here's a quick guide on the various types of firewalls in AWS, along with their purposes:

1. **Security Groups**

- **Purpose**: Acts as a virtual firewall at the **instance level**.

- **Features**:

  - Controls inbound and outbound traffic to EC2 instances.

  - Only allows **stateful** filtering (if inbound is allowed, outbound is automatically allowed).

  - Can only allow traffic (no deny rules).

  - Rules are evaluated on a per-instance basis.

  - Default allows no inbound traffic, allows all outbound traffic.

**Use Case**:

- Protect EC2 instances from unauthorized access by defining rules to allow only specific ports, IP addresses, or protocols.

---

2. **Network Access Control Lists (NACLs)**

- **Purpose**: Acts as a virtual firewall at the **subnet level**.

- **Features**:

  - Controls traffic entering or leaving subnets.

  - **Stateless** filtering (inbound and outbound rules are evaluated separately).

  - Can allow **allow** and **deny** rules.

  - Rules are evaluated based on priority, starting from the lowest numbered rule.

**Use Case**:

- Provides additional security for subnets by controlling traffic flow across entire subnets, suitable for more fine-grained control than security groups.

---

3. **AWS Web Application Firewall (WAF) **

- **Purpose**: Protects **web applications** by filtering and monitoring HTTP/S traffic.

- **Features**:

  - Protects against common web exploits like SQL injection, cross-site scripting (XSS), and bad bots.

  - Rules can be configured to allow, block, or count traffic based on specific conditions like IP addresses, HTTP headers, or request size.

  - Works with AWS services like **CloudFront**, **API Gateway**, and **ALB (Application Load Balancer)**.

**Use Case**:

- Defends against web vulnerabilities at the application layer (Layer 7) for services like websites or APIs.

---

4. **AWS Network Firewall**

- **Purpose**: Provides scalable protection for **VPC** networks at the **network layer**.

- **Features**:

  - Provides stateful, stateless, and deep packet inspection.

  - Capable of managing ingress and egress traffic across multiple subnets in a VPC.

  - Integrates with AWS Firewall Manager for centralized management of rules and policies.

  - Ideal for more complex networking architectures needing deep network traffic filtering.

**Use Case**:

- Enhance network security in large, complex environments with multiple VPCs or sensitive workloads that need advanced traffic inspection and filtering.

---

5. **AWS Shield**

- **Purpose**: Protection against **DDoS (Distributed Denial of Service)** attacks.

- **Features**:

- Automatically protects AWS resources like **EC2**, **ELB**, **CloudFront**, and **Route 53**.

- Two tiers: **Shield Standard** (free, basic DDoS protection) and **Shield Advanced** (paid, more advanced DDoS mitigation and reporting).

- **Shield Advanced** provides 24/7 access to the AWS DDoS Response Team (DRT) and additional cost protections.

**Use Case**:

- Protect mission-critical web applications or APIs from distributed attacks that aim to overwhelm services and make them unavailable.

---

6. **AWS Firewall Manager**

- **Purpose**: Centralized management of firewall rules across multiple accounts and resources.

- **Features**:

  - Easily deploy WAF, Security Groups, and Network Firewall policies.

  - Provides policy compliance and security posture assessments across all resources.

  - Ideal for managing firewalls in large AWS organizations with many accounts and regions.

**Use Case**:

- Ensure consistent security policies across multiple accounts and VPCs, particularly in enterprise environments with centralized governance.

---

Summary of Use Cases:

- **Security Groups**: Instance-level firewall for EC2.

- **NACLs**: Subnet-level firewall for additional control.

- **AWS WAF**: Application-layer protection for web apps.

- **AWS Network Firewall**: Deep inspection of network traffic.

- **AWS Shield**: DDoS protection for critical resources.

- **AWS Firewall Manager**: Centralized management of firewall policies across AWS accounts.

These firewalls provide layers of protection at different levels, from the instance and subnet to the application and network levels.

ASG:

The main difference between vertical and horizontal scaling is how a system's hardware specifications are increased to improve scalability:

- Vertical scaling

  Also known as "scaling up", this method increases the hardware configuration of a server without changing the logical unit. This is done by adding more resources to a single server, such as faster CPUs, more memory, or more storage.

- Horizontal scaling

  Also known as "scaling out", this method increases a system's capacity by adding more instances to the system. This distributes the workload across multiple devices, allowing for parallel execution.

  Here are some other differences between vertical and horizontal scaling:

- Cost

  Vertical scaling often requires more powerful and expensive hardware, while horizontal scaling can use commodity hardware, which can be more cost-effective.

- Scalability

  Vertical scaling limits how much a single server can handle, while horizontal scaling allows for virtually unlimited scalability.

- Flexibility

  Vertical scaling requires downtime or service interruption during hardware upgrades, while horizontal scaling can be done without downtime.

- Fault tolerance

  Vertical scaling relies on a single server, so the entire system may go down if that server fails, while horizontal scaling can still operate if one server fails.

## difference between **serverless** and **server-based computing**

The key difference between **serverless** and **server-based computing** (also known as server computing) lies in how they handle infrastructure management and scaling, and the responsibility of the developer versus the cloud provider.

### 1. **Server Computing (Traditional)**

- **Overview**: In server-based computing, you explicitly manage servers (virtual or physical) that run your applications. This includes provisioning, configuring, and maintaining the servers.

- **Infrastructure**: You are responsible for managing the infrastructure—such as CPU, RAM, disk space, and network settings—whether on-premises or in the cloud (e.g., AWS EC2).

- **Scaling**: Scaling is typically manual or semi-automated. You have to set up auto-scaling, load balancers, and other mechanisms to handle varying loads.

- **Cost**: You pay for the server infrastructure, which often includes idle time when the servers are not fully utilized. Billing is based on uptime or the provisioned capacity, even if not in use.

- **Use Cases**: Applications with steady workloads or where full control over the infrastructure is required, such as enterprise applications, databases, and legacy applications.

**Example Services**:

- **AWS EC2 (Elastic Compute Cloud)**: Virtual servers in the cloud that you manage.

- **AWS RDS (Relational Database Service)**: Managed relational databases that still involve some infrastructure provisioning.

### 2. **Serverless Computing**

- **Overview**: In serverless computing, the cloud provider manages the infrastructure. You focus only on writing code, and the cloud provider automatically handles provisioning, scaling, and maintenance of servers.

- **Infrastructure**: Completely abstracted from the developer. You don’t need to manage the server—it's fully handled by AWS or another cloud provider. You only deploy the code, and the service runs it.

- **Scaling**: Automatically scales based on the load (requests). You don't need to worry about infrastructure resources or auto-scaling configurations.

- **Cost**: You pay only for the actual usage (e.g., number of requests, execution time, storage), rather than paying for the underlying server infrastructure. This can lead to significant cost savings for applications with variable or unpredictable traffic.

- **Use Cases**: Applications with unpredictable traffic, microservices architectures, event-driven applications, and real-time data processing.

**Example Services**:

- **AWS Lambda**: A serverless compute service where you run code in response to events (triggers) without managing the servers.

- **AWS API Gateway**: A serverless API management service that scales automatically.

- **AWS DynamoDB**: A serverless NoSQL database that automatically scales based on demand.

### **Key Differences:**

| Aspect | Server Computing | Serverless Computing |
|---|---|---|
| **Infrastructure Management** | You manage the servers (e.g., EC2 instances, networking, security). | Managed by the cloud provider (e.g., AWS Lambda, DynamoDB). |
| **Scaling** | Manual or auto-scaling, but requires configuration and monitoring. | Automatically scales with demand, no manual intervention. |
| **Cost** | Charged based on provisioned capacity (even during idle times). | Charged based on actual usage (execution time, number of requests). |
| **Responsibility** | You are responsible for server provisioning, patching, scaling, etc. | The cloud provider manages servers, provisioning, and scaling. |
| **Use Cases** | Ideal for steady, predictable workloads or applications requiring full control. | Ideal for applications with variable traffic, microservices, or event-driven tasks. |
| **Startup Time** | Depends on server type (may involve boot time). | Instantaneous execution (near-zero cold start time in most cases). |
| **Development Focus** | You need to handle infrastructure and application logic. | You focus only on application code, infrastructure is abstracted. |

### **Examples**:

- **Server-Based (EC2)**: You deploy an application on an EC2 instance, and you're responsible for ensuring that the instance is running, managing updates, scaling it, and configuring security.

- **Serverless (Lambda)**: You write a function, upload it to AWS Lambda, and AWS automatically handles everything else, including scaling the number of instances required to handle varying loads.

In summary, **server computing** provides more control and flexibility over the infrastructure, while **serverless computing** emphasizes agility and automatic infrastructure management, ideal for dynamic or event-driven applications.