Chapter 5: Elastic Load Balancing, Amazon CloudWatch, & Auto Scaling

One of the biggest selling points for cloud computing is automatic scalability of infrastructure

This chapter covers the technologies that make this possible at AWS

What do we need to implement an automatically scaling infrastructure?

- A mechanism to distribute load across multiple EC2 instances
- A mechanism to monitor utilization, so that we know WHEN more capacity is needed, or when excess capacity can be released
- A mechanism to start/stop EC2 instances

Amazon's solution for each of these is:

- Distribute load: **AWS Elastic Load Balancing (ELB)**
- Monitor utilization: **AWS CloudWatch**
- Start/stop instances: **AWS Auto Scaling**

Elastic Load Balancing

What Is It?

Mechanism that distributes network traffic across multiple server/EC2 instances

Two ways to implement load balancing at AWS:

- Using a custom AMI (bad!)
- Using Amazon's **Elastic Load Balancer (ELB)** service (good!)

ELB is a highly available (within a single region), AWS-managed load balancer service

ELB balances load across EC2 instances in **Availability Zones (AZs)** *within* a single region; it cannot load-balance *across* regions (can use AWS Route 53 to route users to closest region and load balance within)

ELBs are created and managed thru the EC2 menu in the AWS console

ELB Types and Listeners

Three types of ELB load balancers: **Internet-facing**, **internal**, and **HTTPS**

- **Internet-facing** – ELB is accessible from the Internet, used to balance load across web servers
- **Internal** – ELB is within VPC, balances incoming requests across application or data servers
- **HTTPS** – ELB supports SSL encryption (contains cert, acts as endpoint/terminator for SSL)

AWS generates and assigns each ELB a DNS name to address it

- The ELB's IP address is managed by AWS and can change over time
- **You can't assign an Elastic IP address to an ELB**
- Customers can create other DNS names (CNAME recs) to alias the AWS-generated name

A ***listener*** tells the ELB what protocol(s) and port(s) to listen for requests, on the front and back end

- ELB supports these protocols: **HTTP**, **HTTPS**, **TCP** and **SSL**
- ELBs within a VPC support IPv4 only; ELBs in EC2-Classic support IPv4 and IPv6 (aka "**dualstack**")
- SSL requires installation of certificate on the ELB
- Cert must include **Subject Alternative Name (SAN)** for each domain if hosting multiple domains (e.g. sales.coke.com, marketing.coke.com, finance.coke.com; or wildcards, like *.coke.com)

ELB Configuration Settings

Idle Connection Timeout

- ELB maintains two network connections per request: client to ELB, and ELB to back end
- Idle Connection Timeout = how long the ELB will keep an idle connection open before closing
- **Default is 60 seconds**, for both front and back end connections
- Enable keep-alive on EC2 web servers for better performance (reuses connections)
- Set keep-alive timeout on web servers higher than Idle Connection Timeout

Cross-zone Load Balancing

- Ensures requests are balanced across Availability Zones (AZs)
- Helps overcome issue where clients cache DNS lookups and skew toward one AZ

Connection Draining

- Allows unhealthy or deregistering instance a period of time to complete in-flight requests
- Keeps existing connections open
- Stops sending new requests to the instance
- **Timeout value can be 1-3600 seconds**
- **Default is 300 seconds** (5 minutes)

Proxy Protocol

- Allows ELB to inform the back end that a load balancer proxied the incoming request
- Inserts an HTTP header containing source/destination IP and ports
- Multiple ELBs with Proxy Protocol enabled will insert multiple headers

Sticky Sessions

- By default, load balancer routes each request to EC2 instance with lightest load
- Sticky sessions (aka Session Affinity) routes a client's requests to the same instance
- ELB can use session cookie provided by the app to make session sticky
- Alternatively, ELB can insert an **AWSELB cookie** to do same

Health Checks

- Tests status of the EC2 instances behind the load balancer
- Status can be either: **InService** or **OutOfService**
- Three types of health checks: **ping**, **connection attempt**, **web page**
- Configure: **health check type**, **test interval**, **# attempts** before flagging healthy/unhealthy

Amazon CloudWatch

What Is It?

A highly available (within a region), AWS-managed service to monitor AWS resources in real-time

- Collects **metrics** from AWS resources (EC2 instances, ELBs, RDS instances, etc.)
- Uses **alarms** to send **notifications** when a metric meets the criteria
    - For example, CPU utilization > 70%, free disk space < 10%, etc.
- Each alarm watches a single metric and sends a notification when the metric breaches a specified threshold

Two types of monitoring:

- **Basic** (default): Free, limited number of metrics, collects data every five minutes
- **Detailed**: $, larger choices of metrics, collects data every minute, can aggregate over time

Apps can add/inject custom metrics into CloudWatch via an API

- OS-specific metrics that aren't visible to CloudWatch
- Metrics generated by the application itself

Limit of 5000 alarm definitions per account

AWS CloudWatch Logs collects log files from AWS infrastructure

Metrics data is kept for two weeks; move to S3 or Glacier for longer-term storage

Can install a **CloudWatch Logs agent** on Linux servers to push OS or app logs to CloudWatch

Auto Scaling

What Is It?

A highly available service (within a region) that scales your Amazon EC2 capacity automatically, using criteria that you define

Auto Scaling Plans

- **Maintain Current Instance Levels** – Ensure fixed number of instances are always available
- **Manual Scaling** – Administrator scales by changing max/min capacity settings
- **Scheduled Scaling** – Scale on specific dates and/or times
- **Dynamic Scaling** – Scale based on parameters you specify

Auto Scaling Components

- **Launch Configuration** – Template that determines what kind of EC2 instances to create
- **Auto Scaling Group** – Collection of instances managed by Auto Scaling
- **Scaling Policy** (optional) – Specifies how quickly/aggressively to scale out/in

Launch Configuration

- Template that Auto Scaling uses to create new EC2 instances
- Contains:
    - **Configuration Name** – Name you use to refer to the configuration
    - **AMI Name** – Name of the image to use to create the EC2 instance
    - **EC2 Instance Type** - Virtual Machine resource config (memory, CPU, storage, network)
    - **Security Group** – Stateful firewall for the EC2 instances
    - **Instance Key Pair** – Encryption key pair for connection and authentication
    - **Max Bid Price** – Maximum bid price, if using Spot instances
- Can use **On Demand** or **Spot** instances only; can't mix within a single launch configuration
- **Limited to 100 launch configurations per account, per region**
- Consider using a preconfigured AMI versus post-launch config (faster to spin up)
- Can use a new launch configuration to roll out patches/updates to AMIs
    - Auto Scaling will gradually terminate old instances and replace with new ones

Auto Scaling Group

- Collection of EC2 instances managed by the Auto Scaling service
- Contains:
    - **Name** – Name of the Auto Scaling Group
    - **Minimum number of instances** – Lower limit
    - **Maximum number of instances** – Upper limit
    - **Desired number of instances** (optional) – Defaults to minimum

Scaling Policy

- A set of instructions that tells Auto Scaling whether to scale out/in, and how
- Ways to Configure Scaling Policy

    - Increase/decrease by a specific # of instances or by a percentage
    - Increase/decrease based on the size of the alarm threshold trigger
- Can associate more than one Scaling Policy with an Auto Scaling Group
    - To scale based on CPU utilization OR memory usage, for instance
- Best practice is to **scale out quickly, and scale in slowly**