MODULE 10 · STUDY NOTES

Automatic Scaling and Monitoring

Elastic Load Balancing, Amazon CloudWatch, and Amazon EC2 Auto Scaling — the trio that lets an architecture match demand

A modern high-traffic site must serve hundreds of thousands — even millions — of concurrent requests, and demand rarely stays flat. Provision for the peak and most of your fleet sits idle; provision for the trough and the application slows or goes down. This module presents AWS's answer as three services that are useful alone and formidable together: **Elastic Load Balancing** spreads traffic across healthy targets in one or more Availability Zones, **Amazon CloudWatch** watches metrics in real time and raises alarms, and **Amazon EC2 Auto Scaling** launches or terminates instances to match conditions you define. The module closes with Lab 6, where you wire all three into one dynamically scalable architecture.

LEARNING OBJECTIVES

- Indicate how to distribute traffic across Amazon Elastic Compute Cloud (Amazon EC2) instances by using Elastic Load Balancing
- Identify how Amazon CloudWatch enables you to monitor AWS resources and applications in real time
- Explain how Amazon EC2 Auto Scaling launches and releases servers in response to workload changes
- Perform scaling and load balancing tasks to improve an architecture

10.1 Elastic Load Balancing

Elastic Load Balancing (ELB) distributes incoming application or network traffic across multiple **targets** — EC2 instances, containers, IP addresses, and Lambda functions — in a single Availability Zone or across multiple Availability Zones. The load balancer itself is elastic: it scales as traffic to your application changes over time and can automatically scale to most workloads. ELB comes in three types, distinguished by the OSI layer they operate at.

TABLE 10.1 The three load balancer types

TYPE	OSI LAYER	ROUTES	HIGHLIGHTS
Application Load Balancer	7 — application	By **content of the request**; HTTP and HTTPS	Advanced request routing for modern architectures — microservices and container-based apps; targets include EC2, containers, IPs, and **Lambda functions**; always uses the latest SSL/TLS ciphers and protocols
Network Load Balancer	4 — transport	Connections by **IP protocol data**; TCP and UDP	Millions of requests per second at **ultra-low latency**; optimized for sudden, volatile traffic patterns; targets include EC2 instances, microservices, containers
Gateway Load Balancer	3 — network	Traffic to **virtual appliances**	Deploy, scale, and manage firewalls, intrusion detection/prevention, deep-packet-inspection systems; a transparent network gateway — single entry and exit point — that scales appliances with demand

How it works. A load balancer accepts incoming traffic from clients and routes requests to its registered targets in one or more Availability Zones. With all three types you register targets in **target groups** and route traffic to the target groups. You configure the load balancer to accept traffic by specifying one or more **listeners** — a listener is a process that checks for connection requests, configured with a protocol and port for client→load-balancer connections and again for load-balancer→target connections. You can also configure **health checks**: the load balancer sends requests only to healthy instances, stops routing to a target it detects as unhealthy, and automatically resumes when the target is healthy again.

WHY USE A LOAD BALANCER

- **High availability and fault tolerance** — traffic is balanced across healthy targets in multiple AZs; if one AZ's targets fail, traffic shifts to healthy targets in other AZs and returns automatically on recovery.
- **Containerized applications** — load balance across multiple ports on the same EC2 instance; deep Amazon ECS integration manages Docker container registration/deregistration transparently.
- **Automatic scaling** — works with CloudWatch and EC2 Auto Scaling: an alarm (e.g. on instance latency) triggers scaling, new instances are registered, and the LB directs traffic to them.
- **Inside your VPC** — serve as a public entry point or route between application tiers; security groups control open ports, and network ACLs plus route tables still apply. A load balancer is **public by default** or can be **internal** (no internet gateway needed; private IPs in its DNS record).
- **Hybrid load balancing** — one load balancer (or two with DNS-weighted routing) can distribute traffic across AWS **and on-premises** resources registered in target groups.
- **Lambda over HTTP(S)** — register Lambda functions as ALB targets and use content-based routing rules; one ALB can front applications built from servers *and* serverless functions.

WORKED ACTIVITY – PICK THE RIGHT LOAD BALANCER

Traffic to a **containerized application** → *Application* LB. Extremely **spiky, unpredictable TCP** traffic → *Network* LB. **Virtual appliances** for security inspection and traffic analysis → *Gateway* LB. A **static or Elastic IP**, or an IP target outside a VPC → *Network* LB. **Millions of requests per second** at low latency → *Network* LB. **HTTPS** requests → *Application* LB.

TABLE 10.2 Monitoring your load balancer

TOOL	WHAT IT CAPTURES
CloudWatch metrics	ELB publishes data points for load balancers and targets; metrics are ordered time-series data. Verify performance and alarm when a metric leaves the acceptable range (e.g. send an email notification)
Access logs	Detailed information about **requests made to the load balancer**, stored as log files in S3 — analyze traffic patterns, troubleshoot targets and backend apps
AWS CloudTrail logs	Detailed information about **calls made to the ELB API**, stored in S3 — who made the call, what call, when, and from which source IP

COMMON PITFALL

Layer confusion is the classic miss: **A**pplication = layer **7** (HTTP/HTTPS, content-based routing), **N**etwork = layer **4** (TCP/UDP, extreme throughput), **G**ateway = layer **3** (appliances). And it's the *Network* Load Balancer — not the ALB — that supports a static/Elastic IP and handles millions of requests per second.

10.2 Amazon CloudWatch

To use AWS efficiently you need insight into your resources: *When should you launch more EC2 instances? Is performance or availability suffering from a lack of capacity? How much of your infrastructure is actually being used?* **Amazon CloudWatch** answers these questions. It is a monitoring and observability service built for DevOps engineers, developers, site reliability engineers (SREs), and IT managers, and it monitors your AWS resources — and the applications you run on AWS — **in real time**. CloudWatch collects and tracks **metrics**: variables you can measure for your resources and applications, both AWS-standard and **custom** metrics from your own apps and infrastructure. There is no upfront commitment or minimum fee — you pay at the end of the month for what you use.

Alarms. An alarm watches a single CloudWatch metric or the result of a math expression over metrics, and can be based on a **static threshold**, **anomaly detection**, or a **metric math expression**. With a static threshold, the alarm goes to `ALARM` state when the metric breaches the threshold for a specified number of evaluation periods. Alarm actions can **send a notification to an Amazon SNS topic** or **perform an EC2 Auto Scaling or EC2 action**. You can alarm on EC2 CPU utilization, ELB request latency, DynamoDB table throughput, SQS queue length — even the charges on your AWS bill.

TABLE 10.3 Anatomy of a static-threshold alarm

YOU SPECIFY	MEANING / OPTIONS
Namespace	Container of the metric you want — e.g. `AWS/EC2`
Metric	The variable to measure — e.g. CPU utilization
Statistic	Average, sum, minimum, maximum, sample count, or a predefined/custom percentile
Period	Evaluation period; each period is aggregated into one data point
Conditions	Greater, greater-or-equal, lower-or-equal, or lower than a **threshold value** you set
Additional configuration	How many data points within the evaluation period must breach, and how to treat **missing data**
Actions	Notify an Amazon SNS topic · EC2 Auto Scaling action · EC2 action

> **WORKED ACTIVITY – VALID OR BROKEN ALARM?**
>
> An alarm that fires when average CPU is **> 60%** for two periods: *valid*. An alarm on a metric staying **"around" 60%**: *invalid* — "around" is not a threshold option; conditions must use >, >=, <=, or <. An alarm on "volume" with no aggregation: *invalid* — you must specify a **statistic** (e.g. *average* volume).

CloudWatch Events. You define rules that match incoming events — changes in your AWS environment — and route them to one or more targets for processing: EC2 instances, Lambda functions, Kinesis streams, ECS tasks, Step Functions state machines, SNS topics, SQS queues, and built-in targets. CloudWatch Events becomes aware of operational changes *as they occur* and can take corrective action — sending messages, activating functions, making changes, capturing state. The net effect of CloudWatch: **system-wide visibility** into resource utilization, application performance, and operational health.

> **COMMON PITFALL**
>
> The alarm does not deliver the alert itself. Its notification action publishes to an **Amazon SNS topic**, and SNS sends the notification. The module's sample exam question hinges on exactly this: *send alerts based on CloudWatch alarms* → **Amazon SNS** (not CloudTrail, Trusted Advisor, or Route 53).

10.3 Amazon EC2 Auto Scaling

Scaling is the ability to increase or decrease the compute capacity of your application. With a workload that varies across the week, provisioning for the busiest day means paying for underutilized resources every other day; provisioning below peak means the application underperforms — or becomes unavailable — when demand spikes. The deck's own example is Amazon.com: November traffic (Black Friday, Cyber Monday) towers over the rest of the year, and a fixed peak-sized fleet would leave **76 percent of resources idle** most of the year. Without scaling, saturated servers can crash and the business loses customer confidence. In the cloud, computing power is a programmatic resource, so scaling can be flexible and automatic.

Amazon EC2 Auto Scaling helps you maintain application availability by automatically adding or removing EC2 instances according to conditions you define. Its fleet-management features detect impaired EC2 instances and unhealthy applications and **replace the instances without your intervention**. It offers several scaling options — manual, scheduled, dynamic (on-demand), and predictive — and dynamic + predictive can be combined to scale faster.

KEY TERMS

Auto Scaling group
A collection of EC2 instances treated as a logical grouping for scaling and management. Its size is set by desired capacity and adjusted manually or automatically.

Minimum / maximum size
Hard bounds — EC2 Auto Scaling is designed never to let the group shrink below the minimum or grow above the maximum.

Desired capacity
The number of instances the group is steered to; e.g. min 1, desired 2, max 4 means policies move the group between 1 and 4, targeting 2.

Scaling out / scaling in
Launching instances is *scaling out*; terminating instances is *scaling in*.

Launch configuration
The instance configuration template — AMI ID, instance type, IAM role, additional storage, one or more security groups, EBS volumes.

HOW EC2 AUTO SCALING FITS TOGETHER – WHAT, WHERE, WHEN

- **What** you scale: the **launch configuration** — the template every new instance is created from.
- **Where** you scale: the **Auto Scaling group**, launched into a subnet within a VPC, with min / max / desired capacity.
- **When** you scale: the scaling options and policies you specify.
- EC2 Auto Scaling integrates with ELB — attach one or more load balancers to the group and instances are **registered automatically** and receive traffic.

TABLE 10.4 Scaling options

OPTION	HOW IT WORKS	BEST FOR
Maintain current levels	Periodic health checks on running instances; an unhealthy instance is terminated and a new one launched	Keeping a fixed number of instances always running
Manual	You change the minimum, maximum, or desired capacity yourself	One-off adjustments
Scheduled	Scaling actions run automatically as a function of **date and time**, via a scheduled action	Predictable patterns — e.g. traffic rises every Wednesday, stays high Thursday, falls Friday
Dynamic	Scaling policies with parameters you define — e.g. keep fleet CPU near 50% — typically triggered by a **CloudWatch alarm**; policy type determines how the action is performed	Changing conditions when you don't know *when* they'll change; extra capacity for spikes without idle waste
Predictive	Via **AWS Auto Scaling**: ML models trained on your actual EC2 usage plus billions of AWS-observed data points predict traffic, including daily and weekly patterns	Needs **1 day** of history to start; re-evaluated every **24 hours** to forecast the next **48 hours**; produces a scaling plan; combine with dynamic to scale faster

WORKED EXAMPLE – DYNAMIC SCALING WITH THE TRIO

You create a CloudWatch alarm on fleet CPU utilization: if the average exceeds **60% for 5 minutes**, scaling policies run. The alarm fires → EC2 Auto Scaling instantiates a new instance from your **launch configuration** into the group → Auto Scaling calls ELB to **register** the new instance → ELB runs **health checks**, then starts distributing traffic to it — while feeding its metrics back to CloudWatch. Each service is useful alone; together they close the loop and give you control and flexibility over how the application meets customer demand.

AWS Auto Scaling is a **separate service** from Amazon EC2 Auto Scaling. It monitors your applications and automatically adjusts capacity to maintain steady, predictable performance at the lowest possible cost, through a simple UI for building **scaling plans** across multiple resource types: **EC2 instances and Spot Fleets**, **Amazon ECS tasks**, **DynamoDB tables and indexes**, and **Aurora Replicas**. If you already use EC2 Auto Scaling for your instance fleet, AWS Auto Scaling extends scaling to those other services — and provides the predictive scaling described above.

COMMON PITFALL

Two names, two services. **Amazon EC2 Auto Scaling** launches and terminates EC2 instances in Auto Scaling groups. **AWS Auto Scaling** is the umbrella service for scaling plans across EC2/Spot Fleets, ECS, DynamoDB, and Aurora Replicas — and it is what powers predictive scaling.

10.4 Lab 6 – Scale and Load Balance Your Architecture

The module's lab (~30 minutes) turns the three services into one working architecture: starting from a given infrastructure, you make it load balanced and dynamically scalable.

LAB TASKS

- Create an **Amazon Machine Image (AMI)** from a running instance.
- Create an **Application Load Balancer**.
- Create a **launch configuration** and an **Auto Scaling group**.
- Automatically scale new instances within a **private subnet**.
- Create **CloudWatch alarms** and monitor performance of the infrastructure.

Module summary

- Elastic Load Balancing distributes application or network traffic across targets — EC2 instances, containers, IP addresses, Lambda functions — in one or more Availability Zones, and scales itself with traffic.
- Three types by OSI layer: Application LB (layer 7, HTTP/HTTPS, content-based routing), Network LB (layer 4, TCP/UDP, millions of requests/sec, static IP), Gateway LB (layer 3, virtual appliances).
- Load balancers use listeners (protocol + port), target groups, and health checks — traffic goes only to healthy targets and resumes automatically on recovery.
- Monitor a load balancer with CloudWatch metrics, access logs in S3 (request detail), and CloudTrail logs in S3 (who called the ELB API, what, when, from where).
- CloudWatch monitors AWS resources and applications in real time: standard + custom metrics; alarms based on static threshold, anomaly detection, or metric math; alarm actions notify an SNS topic or perform EC2 Auto Scaling / EC2 actions; Events rules route environment changes to targets.
- Amazon EC2 Auto Scaling keeps applications available by launching (scaling out) and terminating (scaling in) instances: an Auto Scaling group (min / max / desired) launches from a launch configuration, with maintain, manual, scheduled, dynamic, and predictive options.
- Dynamic scaling is the trio in a loop: CloudWatch alarm → EC2 Auto Scaling launches from the launch configuration → instance registers with ELB → ELB health checks and routes traffic → metrics flow back to CloudWatch.
- AWS Auto Scaling is a separate service that builds scaling plans for EC2 instances and Spot Fleets, ECS tasks, DynamoDB tables and indexes, and Aurora Replicas — and powers predictive scaling (1 day of history, 24-hour re-evaluation, 48-hour forecast).

Review questions

1. Your web tier runs in two Availability Zones behind a load balancer, and every instance in one AZ fails its health checks. What happens to user traffic?

Answer: The load balancer stops routing to the unhealthy targets and sends all traffic to healthy targets in the other AZ. When the failed targets pass health checks again, ELB automatically resumes routing traffic to them — no operator action required.

2. Choose a load balancer for each: (a) an HTTPS microservices app in containers, (b) spiky TCP traffic needing a static IP, (c) a fleet of inspection firewalls.

Answer: (a) Application LB — layer 7, content-based routing, container/microservice support, HTTPS. (b) Network LB — layer 4, optimized for volatile traffic, supports static/Elastic IPs and millions of requests per second. (c) Gateway LB — layer 3 transparent gateway that deploys and scales virtual appliances.

3. What must be configured before a load balancer can accept and forward traffic, and what does each piece do?

Answer: At least one listener (protocol + port for client→LB and LB→target connections) to accept traffic, targets registered in target groups to receive it, and — best practice — health checks so requests go only to healthy targets.

4. You need to (a) find out who made a configuration call to the ELB API and (b) analyze the pattern of client requests hitting the load balancer. Which log source answers each?

Answer: (a) AWS CloudTrail logs — who made the call, what call, when, and the source IP, stored in S3. (b) Access logs — detailed information about requests sent to the load balancer, also stored in S3, used for traffic analysis and troubleshooting targets.

5. List everything you must specify when creating a static-threshold CloudWatch alarm.

Answer: Namespace (e.g. AWS/EC2), metric (e.g. CPU utilization), statistic (average, sum, min, max, sample count, or percentile), period, conditions (>, >=, <=, or < plus the threshold value), additional configuration (data points that must breach and missing-data treatment), and actions (SNS notification, EC2 Auto Scaling action, or EC2 action).

6. An Auto Scaling group has minimum 1, desired 2, maximum 4. Explain what each number does.

Answer: EC2 Auto Scaling is designed never to let the group fall below 1 or exceed 4 instances; it steers the group toward the desired 2. Scaling policies move the instance count between the min and max based on the criteria you define.

7. Why would you combine dynamic and predictive scaling, and what does predictive scaling need to work?

Answer: Dynamic scaling reacts to conditions as they change; predictive scaling gets capacity ready ahead of forecast demand — together they scale faster. Predictive scaling uses ML trained on your actual EC2 usage plus AWS's billions of data points; it needs at least 1 day of history, re-evaluates every 24 hours, and forecasts the next 48 hours as a scaling plan.

8. Trace what happens, service by service, when average fleet CPU stays above 60% for 5 minutes under a dynamic scaling policy.

Answer: The CloudWatch alarm fires and triggers the scaling policy → EC2 Auto Scaling launches a new instance from the launch configuration into the Auto Scaling group → Auto Scaling calls ELB to register the instance → ELB health checks it, then routes traffic to it, while continuing to feed metrics to CloudWatch — closing the loop.