MODULE 4 · STUDY NOTES

Computing (Scaling and Name Resolution)

Distributing traffic with Elastic Load Balancing, adjusting capacity with EC2 Auto Scaling, and resolving and routing names with Amazon Route 53

Elastic architectures in AWS are built from three cooperating services, and this module works through each one and how they fit together. In a traditional data center, capacity is bound by the hardware you buy — enough servers to survive the peak, sitting idle the rest of the year. In the cloud, compute is a programmatic resource, so you provision capacity to match demand and pay only for what you use. **Elastic Load Balancing (ELB)** spreads incoming traffic across a fleet of targets so no single instance is overwhelmed. **Amazon EC2 Auto Scaling** grows and shrinks that fleet automatically based on health, CloudWatch alarms, schedules, or machine-learning forecasts. **Amazon Route 53** is the DNS front door that resolves your domain name and routes each user to the right place — the nearest Region, a weighted slice of a blue/green rollout, or a healthy failover target. Together they deliver applications that are highly available, fault-tolerant, and cost-efficient under changing load.

LEARNING OBJECTIVES

- Describe Elastic Load Balancing features
- Differentiate the types of ELB load balancers
- Describe Amazon EC2 Auto Scaling and launch configurations
- Use EC2 Auto Scaling in AWS
- Describe Amazon Route 53 features and routing options
- Configure failover routing

4.1 Scaling on demand

Buying hardware to meet short-term demand creates waste — servers stand idle during off-peak periods and become a constraint during peaks. Consider a US online tax-preparation business: traffic climbs steadily from mid-January and peaks near the April 15 filing deadline. A data center must be provisioned for that four-month spike, then those servers sit idle the rest of the year. In the cloud, because computing power is a **programmatic resource**, you take a more flexible approach: create new EC2 instances ahead of a known peak, monitor critical resources such as average CPU utilization and **scale out** automatically when they become constrained, and **scale in** when demand returns to baseline. In other words, you pay for the resources you need, when you need them.

THREE COMPONENTS BUILD THE SCALABLE ARCHITECTURE

- **Amazon Route 53** — a highly available, scalable cloud DNS web service that translates names (like www.example.com) into IP addresses and routes users to internet applications. Route 53 entries are often configured to point to an ELB load balancer.
- **Elastic Load Balancing (ELB)** — automatically distributes incoming application traffic across multiple targets in one or more Availability Zones. ELB load balancers are often configured to point to Auto Scaling groups.
- **Auto Scaling group** — a collection of EC2 instances treated as a logical unit for scaling and management; it maintains availability and adjusts capacity up or down automatically according to conditions you define.

Each service hands off to the next: DNS resolves to the load balancer, the load balancer spreads traffic across the group, and the group grows or shrinks the fleet. The rest of the module examines the three services in turn — ELB, EC2 Auto Scaling, and Route 53 — and closes with load testing to prove the design works.

4.2 Elastic Load Balancing (ELB)

Elastic Load Balancing is a managed service that distributes incoming application traffic across multiple targets — EC2 instances, containers, IP addresses, and Lambda functions — in a single Availability Zone or across several. Load balancing is a common feature of web application deployments: ELB routes client requests across the instances or ECS containers running your application so no one target is overwhelmed. You configure a load balancer in the Amazon EC2 area of the AWS Management Console, or through the AWS CLI or SDKs, and choose from three types. ELB also provides **cross-zone load balancing**, which balances requests across targets in all enabled Availability Zones; it is enabled by default on Application Load Balancers, uses health checks to shift traffic, absorbs the impact of DNS caching, and carries **no additional bandwidth charge** for cross-zone traffic — improving fault tolerance and evening out back-end utilization.

KEY ELB FEATURES

- **High availability** — distribute traffic across multiple targets in one or multiple Availability Zones.
- **Health checks** — detect unhealthy targets, stop sending them traffic, and spread the load across the remaining healthy targets.
- **Security** — associate security groups with the load balancer, and optionally create an internal (non-internet-facing) load balancer.
- **TLS/SSL termination** — integrated certificate management and SSL decryption centralize your SSL settings and offload CPU-intensive work from your instances.
- **Layer 4 or Layer 7** — balance HTTP/HTTPS applications at Layer 7 (the OSI application layer) or use strict Layer 4 for applications that rely purely on TCP (the transport layer).
- **Operational monitoring** — integration with Amazon CloudWatch metrics and request tracing for real-time performance monitoring.

TABLE 4.1 Differentiating the three load balancer types

TYPE	OSI LAYER	TRAFFIC	IDEAL FOR
Application (ALB)	Layer 7 — application	HTTP, HTTPS; WebSocket, HTTP/2	Advanced/content-based routing, microservices, and containers; flexible application management; supports a static IP; recommended for most VPC use cases
Network (NLB)	Layer 4 — connection	TCP, UDP	Extreme performance and ultra-low latency, millions of requests/sec; one static IP per AZ; sudden, volatile traffic; recommended for TCP/UDP in a VPC
Classic (CLB)	Request and connection level	HTTP, HTTPS, TCP	Previous generation; applications built on the EC2-Classic network

An **Application Load Balancer** operates at the application layer, supports content-based routing, and provides added visibility into the health of target instances and containers — ideal for websites and mobile apps in containers or on EC2. A **Network Load Balancer** routes connections based on IP protocol data and is optimized for high throughput at very low latency, using a single static IP address per Availability Zone. A **Classic Load Balancer** provides basic load balancing at both the request and connection levels and is intended for the older EC2-Classic network; Application and Network Load Balancers are recommended for all other VPC use cases.

KEY TERMS

Listener
A process that checks for connection requests from clients using the protocol and port you specify. Each load balancer needs at least one listener to accept traffic and can have up to 50; routing rules are defined on listeners.

Target group
A group of registered targets — such as EC2 instances or ECS container instances — that a rule routes requests to. Health checks are configured per target group, and a single target can be registered with multiple target groups.

Listener rule
Specifies a target group, a condition, and a priority. When the condition is met, traffic is forwarded to that target group. Each listener must have a default rule; adding rules that route by request content enables content-based routing.

WORKED EXAMPLE – CONTENT-BASED ROUTING FOR EXAMPLE.COM

One Application Load Balancer is the single point of contact for www.example.com and is configured with rules that route by the requested URL. Requests to **/orders** go to one set of EC2 instances, requests to **/images** go to another set, and requests to **/registration** go to a third — the load balancer inspects the path and forwards each to the matching target group. A listener can carry more than one rule, each with a condition and priority, plus a required default rule; a single target can even belong to multiple target groups. This is exactly the Layer 7 behavior a Network Load Balancer cannot provide.

CREATING AN APPLICATION LOAD BALANCER WITH THE AWS CLI

- `create-load-balancer` — create the load balancer; you must specify **two subnets that are not in the same Availability Zone**.
- `create-target-group` — create a target group; you must specify the VPC where your EC2 instances run.
- `register-targets` — register your EC2 instances with the target group.
- `create-listener` — create a listener with a default rule that forwards requests to the target group.
- `describe-target-health` — optionally verify the health of the registered targets. (These are `aws elbv2` commands.)

COMMON PITFALL

The ALB/NLB distinction is a favorite exam swap. Anything about **HTTP/HTTPS, path- or host-based (content) routing, WebSocket, or HTTP/2** points to the **Application Load Balancer at Layer 7**. Anything about **raw TCP/UDP, extreme performance, millions of requests per second, or a static IP per AZ** points to the **Network Load Balancer at Layer 4**. The Classic Load Balancer is only the answer when the scenario names the EC2-Classic network.

4.3 Amazon EC2 Auto Scaling

Amazon EC2 Auto Scaling automatically launches or terminates EC2 instances based on health status checks, user-defined policies driven by Amazon CloudWatch, schedules, other programmatic criteria, or manual changes with the `set-desired-capacity` command. It maintains application availability, lets you **scale out** to meet demand and **scale in** to reduce cost, and is configured in three parts. To set it up you first create a launch configuration, then create an Auto Scaling group that uses it, and then define the scaling policies that control its behavior.

TABLE 4.2 The three parts of EC2 Auto Scaling

COMPONENT	DEFINES	KEY DETAILS
Launch configuration / template	What to launch	AMI, instance type, security group, key pair, storage, IAM roles, and user data. Only one launch configuration is active at a time; it does not set the VPC/subnet — its one network option is whether to auto-assign a public IP
Auto Scaling group	Where and how many	Minimum, desired (optional), and maximum size; distributes and balances instances across Availability Zones; optional ELB integration; uses health checks to maintain group size
Scaling policy	When and how much	Triggered by CloudWatch, an instance health check, a schedule, or manually; changes capacity by `ChangeInCapacity` (a set number), `ExactCapacity` (an exact number), or `ChangeInPercent` (a percentage of desired)

The launch configuration's requirements resemble those of launching an ordinary EC2 instance — except that you do **not** specify a VPC or subnet, because the Auto Scaling group supplies those. A private instance behind a public ELB load balancer does not need a public IP address, which is why the auto-assign option exists.

Instance health. EC2 Auto Scaling tracks the health state of every instance and terminates any marked unhealthy, replacing it to keep the group functioning. By default it uses Amazon EC2 instance status checks; when the group sits behind a load balancer, you can configure it to use the **load balancer's instance checks** or the **EC2 instance checks**. External scripts can force an instance to recycle by invoking the `aws autoscaling set-instance-health` command.

TABLE 4.3 Custom termination policies (which instance is removed on scale in)

TERMINATION POLICY	WHAT IT SELECTS
OldestInstance	The longest-running instance — useful when upgrading the group to a new instance type
NewestInstance	The shortest-running instance — useful when testing a new launch configuration
OldestLaunchConfiguration	The instance with the oldest launch configuration — good for phasing out a previous config (a default)
ClosestToNextInstanceHour	The instance closest to its next billing hour — maximizes usage and manages cost (a default)

When scaling in, the **default termination policy** first targets the Availability Zone with the largest number of instances, keeping the group balanced across AZs for high availability; if you list multiple policies, they run in the order given. The default is sufficient for most situations, but you can replace it with a custom policy, or call the `TerminateInstanceInAutoScalingGroup` API action to terminate a specific instance and optionally decrement the desired group size.

Steady-state groups. Setting the minimum, maximum, and desired size all to **1** creates a steady-state group that keeps exactly one instance running and recreates it automatically if it becomes unhealthy or its Availability Zone fails. A common use is a NAT server in each AZ. Be aware, though: while the failed instance recycles there is still potential downtime, so a single NAT instance built this way remains a single point of failure.

Scaling approaches. *Scheduled scaling* responds to predictable load changes — you create a scheduled action that sets a new minimum, maximum, and desired size at a specific (possibly recurring) time, which is an excellent way to prewarm capacity for an anticipated spike. *Dynamic scaling* reacts to live metrics through three policy types (below). *Predictive scaling* uses machine learning models trained on your actual EC2 usage plus AWS observations to forecast daily and weekly patterns; it needs historical data from **at least one day**, is re-evaluated **every 24 hours** to forecast the **next 48 hours**, and provisions capacity in advance by forecasting load and scheduling minimum capacity. Predictive scaling suits applications with **periodic, cyclic** spikes and pairs well with dynamic target-tracking scaling; it is not designed for spikes that are not cyclic or predictable.

TABLE 4.4 Dynamic scaling policy types

POLICY TYPE	HOW IT SCALES	NOTES
Target tracking	Increases or decreases capacity to keep a metric at a target value — like a thermostat holding a temperature	Recommended when the metric changes proportionally to the number of instances
Step scaling	Adjusts capacity using step adjustments that vary with the size of the alarm breach	Does not support cooldown; uses an instance warmup period instead
Simple scaling	Makes a single scaling adjustment when an alarm fires	Uses a cooldown period between actions

AVOID THRASHING

- **Alarm sustain period** — trigger actions only on sustained state changes, e.g., CPU utilization above 90% for 10 minutes, so brief blips don't cause scaling.
- **Cooldown period** — after a scale-out or scale-in, suspend further scaling activities for a set time (e.g., 5 minutes). Used with **simple scaling** policies; not with target tracking, step, or scheduled scaling.
- **Instance warmup period** — the seconds a newly launched instance takes to warm up (e.g., 5 minutes). Used with **step scaling**: until warmup elapses the instance is not counted in the group's aggregated metrics or as current capacity, so multiple breaches within one step result in a single scaling activity.
- Note: if an instance becomes unhealthy, Auto Scaling does **not** wait for the cooldown period before replacing it.

WORKED EXAMPLE – THE STEP SCALING PREDICTION CHALLENGE

A group starts with **10 instances** and a **5-minute warmup**. The step policy: add 1 instance when average CPU is 60–80%, add 2 when 80–100%, remove 1 when 20–40%, remove 2 when 0–20% — each sustained for more than 2 minutes. Walk the six conditions. (1) CPU holds 63–70% → the add-1 alarm launches one instance, which begins warming. (2) Two minutes later CPU still sits at 60–80%, so the add-1 alarm fires again — but the first instance is still warming, so **nothing new is added**. (3) Before 5 minutes elapse CPU jumps to 85%, triggering add-2; because the warming instance already counts toward capacity, only **one more** launches, bringing the group to **12**. (4) The first added instance finishes warming and shares the load while the second still warms; CPU falls to about 53% — in no alarm range, so **no action** is taken. (5) CPU settles at 32% for 2 minutes → the remove-1 alarm terminates one instance, leaving **11**. (6) CPU drops to about 17% → the remove-2 alarm terminates two, leaving **9**. The lesson: instances that are warming up already count toward capacity, so step scaling never piles on more instances than you need.

Lifecycle hooks let you intervene before an Auto Scaling action completes. On a scale-out event, Auto Scaling can launch the instance and send a notification, then wait for a user or programmed action — such as installing software — before adding it to the group. On a scale-in event, a hook can remove the instance from the group and send a notification, giving you a chance to act — such as retrieving the instance's logs — before it is terminated.

COMMON PITFALL

Keep cooldown and warmup straight. **Cooldown** belongs to **simple scaling** and pauses further scaling after an action. **Warmup** belongs to **step scaling** and delays counting a new instance until it is ready. Step scaling does not support cooldown at all. Likewise, remember the launch configuration does not set the VPC/subnet — the Auto Scaling group does — and only one launch configuration is active at a time.

4.4 Amazon Route 53

ELB and EC2 Auto Scaling give you flexible, scalable, resilient designs within a Region — but distributing traffic **across AWS Regions** for disaster recovery or lower latency calls for DNS. **Amazon Route 53 is a highly available, scalable cloud Domain Name System (DNS) web service** that reliably and cost-effectively routes end users to internet applications by translating names — like www.example.com — into the numeric IP addresses computers use, such as 192.0.2.1. It connects requests to infrastructure running in AWS (EC2 instances, ELB load balancers, S3 buckets) and to resources outside AWS, supports **DNS health checks** so it routes traffic only to healthy endpoints, manages traffic globally through a variety of routing policies that can combine with DNS failover, and also offers **domain name registration** — you can purchase and manage domains such as example.com and let Route 53 configure the DNS settings automatically.

Associating DNS names with an ELB load balancer. When you create a load balancer, AWS assigns it a default DNS name that resolves to a set of IP addresses (for example, web-app.us-west-2.elb.amazonaws.com). You can use that default, or associate your own domain name — managed in Route 53 — using an alias resource record set or a CNAME record that points to the load balancer.

TABLE 4.5 Alias record vs. CNAME record

RECORD	POINTS TO	NOTES
Alias	Selected AWS resources only — an ELB load balancer, a CloudFront distribution, an S3 bucket, or another record in the same Route 53 hosted zone	AWS-specific; used to map your domain to an ELB's assigned name
CNAME	Any DNS record, whether an AWS or non-AWS name	Redirects one name to another, e.g., apex.example.com to acme.example.org

TABLE 4.6 The seven Route 53 routing policies

ROUTING POLICY	WHEN TO USE IT
Simple	A single resource that performs a given function for your domain — for example, one web server serving example.com
Weighted	Route traffic to multiple resources in proportions you specify — the basis of blue/green traffic shifting
Latency	You have resources in multiple AWS Regions and want to route to the Region with the lowest latency
Failover	Configure active-passive failover to a standby resource
Geolocation	Route based on the location of your users (where the DNS query originates)
Geoproximity	Route based on the location of your resources; optionally shift (bias) traffic from one location to another
Multivalue answer	Respond to DNS queries with up to eight healthy records selected at random

Latency-based routing (LBR) distributes a deployment across several Regions and gives users the fastest response. You create a latency resource record set for each load balancer; when a user in Barcelona enters your domain, DNS routes the request to a Route 53 name server, which consults its latency data and routes to the best-performing Region. Often — but not always — that is the geographically nearest location: Australia for a user in New Zealand, or Oregon for a user in Canada.

> **WORKED EXAMPLE – BLUE/GREEN DEPLOYMENT WITH WEIGHTED ROUTING**
>
> A blue/green deployment runs two matching production environments to reduce the risk of an outage during a release: the existing **blue** environment and the new **green** environment, each with its own ELB load balancer and EC2 Auto Scaling configuration. Route 53 **weighted routing** begins shifting users from blue to green — for example moving from 100/0 to 95/5 and onward — while CloudWatch and CloudWatch Logs monitor the green side. If problems appear, weighted routing shifts users **back to blue**; once green runs cleanly, blue is gradually shut down. Because DNS records are cached, a full shutdown of blue can take anywhere from a day to a week.

TESTING SCALABILITY

- Test your scalability deployments so you find problems earlier rather than later — before they reach production.
- Simulate traffic with **load testing** to confirm the design scales as expected.
- Common open source tools include **The Grinder**, **Apache JMeter**, and **Bees with Machine Guns** — a simple load-testing script specific to Amazon EC2.

COMMON PITFALL

Geolocation vs. geoproximity trips people up. **Geolocation** routes on the location of your **users** — where the DNS query comes from. **Geoproximity** routes on the location of your **resources** (an AWS Region, or latitude and longitude) and can bias traffic between locations. And remember: **failover** routing gives active-passive; **weighted** routing (not failover) is what powers gradual blue/green traffic shifts.

Module summary

- A scalable, on-demand architecture combines three services: Route 53 resolves the domain and points to an ELB load balancer, ELB distributes traffic across targets, and an EC2 Auto Scaling group grows or shrinks the fleet — so you pay for capacity only when you need it.
- ELB is a managed service that distributes traffic across EC2 instances, containers, IP addresses, and Lambda functions in one or many AZs, with high availability, health checks, security groups, TLS termination, Layer 4/7 balancing, and CloudWatch monitoring; cross-zone balancing is on by default for ALBs.
- The three load balancer types: Application (ALB, Layer 7, HTTP/HTTPS, content-based routing), Network (NLB, Layer 4, TCP/UDP, extreme performance, static IP per AZ), and Classic (CLB, previous generation, EC2-Classic).
- ELB routing is built from listeners (up to 50 per LB, at least one required), target groups (registered EC2/ECS targets with per-group health checks), and listener rules (target group + condition + priority, plus a default rule).
- EC2 Auto Scaling has three parts — launch configuration/template (what to launch), Auto Scaling group (where and min/desired/max), and scaling policy (when and how much) — and scales on health, CloudWatch alarms, schedules, or predictive forecasts.
- Scaling policy types: dynamic (target tracking, step, simple), scheduled, and predictive; simple scaling uses a cooldown period while step scaling uses an instance warmup period and does not support cooldown; warming instances count toward capacity to prevent thrashing.
- Amazon Route 53 is a highly available cloud DNS service that resolves names to IPs, registers/transfers domains, runs DNS health checks, and routes globally; alias records point to AWS resources while CNAME records point to any DNS name.
- Route 53 supports seven routing policies — simple, weighted, latency, failover, geolocation, geoproximity, and multivalue answer; latency-based routing picks the fastest Region, failover gives active-passive, and weighted routing powers blue/green deployments. Load-test the design to confirm it scales.

Review questions

1. **How do Route 53, ELB, and an Auto Scaling group work together in an on-demand architecture?**

   Answer: Route 53 (DNS) resolves the domain name and points to an ELB load balancer; the load balancer distributes incoming traffic across healthy targets; and the EC2 Auto Scaling group launches or terminates instances across Availability Zones to match demand, scaling out to meet load and in to save cost.

2. **Differentiate the ALB, NLB, and CLB by layer and use case.**

   Answer: The Application Load Balancer works at Layer 7 (HTTP/HTTPS) and does content-based routing for microservices and containers. The Network Load Balancer works at Layer 4 (TCP/UDP) for extreme performance and ultra-low latency, with one static IP per AZ. The Classic Load Balancer is previous-generation, operating at both request and connection levels, for the EC2-Classic network.

3. **What is defined in a launch configuration, and what is deliberately left out?**

   Answer: It defines the instance blueprint — AMI, instance type, security group, key pair, storage, IAM roles, and user data — and can set whether to auto-assign a public IP. It deliberately omits the VPC and subnet, which are specified by the Auto Scaling group; only one launch configuration is active at a time.

4. **Which scaling policies use a cooldown period, and which use an instance warmup period?**

   Answer: Simple scaling policies use a cooldown period to pause further scaling after an action. Step scaling policies do not support cooldown; they use an instance warmup period, during which a new instance is not counted in the group's metrics or current capacity.

5. **When is predictive scaling the right choice, and what are its data requirements?**

   Answer: Predictive scaling suits applications with periodic, cyclic traffic spikes. It uses machine learning on daily and weekly patterns, needs at least one day of historical data to start, and is re-evaluated every 24 hours to forecast the next 48 hours; it pairs well with dynamic target-tracking scaling.

6. **What is the difference between an alias record and a CNAME record in Route 53?**

   Answer: An alias record can point only to selected AWS resources — an ELB load balancer, a CloudFront distribution, an S3 bucket, or another record in the same hosted zone. A CNAME record can redirect a DNS query to any DNS record, AWS or not.

7. **A team wants to gradually shift users from an old environment to a new one and roll back if problems appear. Which Route 53 routing policy fits, and why not failover?**

   Answer: Weighted routing — it sends traffic to multiple resources in proportions you specify, so you can move a small percentage to the new (green) environment, monitor it, and shift back to blue if needed. Failover routing is active-passive; it switches to a standby on failure rather than splitting traffic by percentage.