MODULE 10 · STUDY NOTES

Implementing Monitoring, Elasticity, and High Availability

Monitoring with CloudWatch and EventBridge, elasticity through EC2 Auto Scaling and database scaling, and high availability with Elastic Load Balancing and Route 53

Earlier modules assembled the pieces of an architecture; this module makes that architecture *perform* — staying responsive, elastic, and available as load rises, components fail, and whole Availability Zones or Regions go dark. It opens with **monitoring**, because you cannot react to what you cannot see: Amazon CloudWatch gathers metrics and logs, raises alarms on sustained threshold breaches, and Amazon EventBridge routes the resulting events to the services that act on them. Those alarms feed **elasticity** — Amazon EC2 Auto Scaling adds and removes instances across Availability Zones through scheduled, dynamic, and predictive policies — and the same idea extends to **databases**, where Aurora, Aurora Serverless, Amazon RDS, and DynamoDB each scale vertically, horizontally, or both. **High availability** then removes single points of failure: Elastic Load Balancing spreads traffic over healthy targets in multiple AZs, and Amazon Route 53 provides DNS-level failover across Regions. The module closes by mapping all of it back to the reliability and performance-efficiency pillars of the **AWS Well-Architected Framework**. Throughout, the cloud architect works backward from the business need — availability targets, traffic shape, and cost — to choose the right combination.

LEARNING OBJECTIVES

- Examine how reactive architectures use Amazon CloudWatch and Amazon EventBridge to monitor metrics and facilitate notification events
- Use Amazon EC2 Auto Scaling within an architecture to promote elasticity and create a highly available environment
- Determine how to scale your database resources
- Identify a load balancing strategy to create a highly available environment
- Use Amazon Route 53 for DNS failover
- Use the AWS Well-Architected Framework principles when designing highly available systems

10.1 Monitoring your resources

A responsive system means that distributed components respond in a timely manner even under heavy load, which puts special attention on **latency**. Synchronous, distributed services often have to respond within a fixed time, so when something goes wrong you need a low-latency error-handling mechanism to report the problem and take remedial action. A distributed application that passes messages between services is also hard to debug, because you cannot always reproduce the exact state of the whole application and replay events. The solution is to consolidate each service's separate log file into a **central service** that provides a complete log of the whole application, aggregate metrics from those logs, and set **alarms** for exceeded thresholds.

THREE CATEGORIES OF METRICS TO MONITOR

- **Operational health** — monitors the application infrastructure so the runtime environment stays healthy.
- **Resource utilization** — measures use of application components so they are neither over- nor under-utilized.
- **Application performance** — measures whether application demand is being met.

Amazon CloudWatch is the AWS metrics and logs repository. An AWS service such as Amazon EC2 *puts* metrics into the repository, and you *retrieve* statistics — aggregations of metric data points over specified time periods — from it; you can put your own **custom metrics** in as well. Metrics are stored separately per Region, but CloudWatch cross-Region functionality can aggregate statistics from different Regions, and a console **dashboard** graphs the results. CloudWatch also queries data from multiple sources, giving visibility across hybrid and multicloud metrics in a single dashboard view. Its **alarms** provide the trigger for responsive, event-driven architectures, and its notifications can drive changes to monitored resources.

KEY TERMS

Metric
A time-ordered set of data points with a unit of measure (bytes, seconds, count, percent) — a variable to monitor, with the data points as its values over time

Standard vs. high resolution
Standard-resolution metrics have 1-minute granularity; high-resolution metrics have 1-second granularity

Namespace
A container that isolates metrics so metrics from different applications are not aggregated together; AWS namespaces use the AWS/service naming form

Dimension
A name-value pair specifying a metric characteristic (for example, InstanceId and the actual instance ID) used to filter when searching

CloudWatch Logs centralizes log files from sources such as EC2 instances, AWS CloudTrail, Route 53, Amazon VPC, and other AWS services into a single, highly scalable service. You can view logs, search them for error codes or patterns, filter them by field, or archive them securely — seeing all logs, regardless of source, as one consistent flow of events ordered by time. CloudWatch Logs also supports a powerful query language, auditing and masking of sensitive data, and generating metrics from logs by using filters or an embedded log format.

A **CloudWatch alarm** watches a single metric over a specified period and performs one or more actions based on the value of the metric relative to a threshold **over time**. The action is a notification to an Amazon SNS topic or to an Auto Scaling policy. Crucially, alarms invoke actions only for **sustained state changes** — the state must have changed and been maintained for a specified number of periods. You control the comparison: the period, and how many **evaluation periods** are used to reach a conclusion. If you set an alarm on a high-resolution metric, you can use a high-resolution alarm with a 10-second or 30-second period (at a higher charge) or a regular alarm with any multiple of 60 seconds.

WORKED EXAMPLE – AN ALARM THAT SCALES OUT

Configure an alarm to activate when the **CPUUtilization** metric is greater than 70 percent for 5 minutes. When that sustained breach occurs, the alarm invokes an action — such as adding an instance through the Auto Scaling policy, or sending a notification to the development team's SNS topic. If you specify three evaluation periods, CloudWatch compares a window of three data points and notifies you only if the oldest data point is breaching and the others are breaching or missing. A metric that breaks a threshold every second should not be acted on every second, which is why metrics are aggregated over time so that one action can solve the problem.

CloudWatch dashboards are customizable home pages in the CloudWatch console that monitor resources in a single view, even resources spread across different Regions; you can also view automatic dashboards. You choose the color for each metric so you can track the same metric across multiple graphs, build a common view of critical measurements to speed communication during operational events, and create an operational playbook for how teams respond to specific incidents. If you prefer not to use the console, you can query CloudWatch statistics to import the data into a custom dashboard.

When an alarm is breached, the event needs to be published so that automated or manual correction can happen. **Amazon EventBridge** is a serverless service that uses events to connect application components, enabling loosely coupled, event-driven architectures. It processes events with **event buses** and **pipes**, and makes routing decisions with configurable **rules** that match on an event pattern or run on a schedule (through Amazon EventBridge Scheduler). Targets can be in another AWS account, in another Region, or both.

EVENT BUS	PIPE (EVENTBRIDGE PIPES)
many sources → many targets ▪ A router that receives events and delivers them to zero or more targets ▪ Use to broker between workloads, services, or systems ▪ Multiple buses can divide traffic (for example, PII vs. non-PII) or aggregate into a centralized bus ▪ A single rule can fan an event out to targets that run in parallel	*one source → one target* ▪ Intended for point-to-point integrations ▪ Receives events from a single source for delivery to a single target ▪ Supports advanced transformation and enrichment before delivery ▪ Simpler than wiring a bus and rule for a 1:1 flow

A **rule** matches incoming events and sends them to targets for processing; you can define up to **five targets** per rule, and EventBridge supports many AWS services and API destinations. An **event pattern** contains one or more filters that define the event structure and the fields to match, while a **schedule** invokes targets at regular intervals. For example, an event pattern that captures every EC2 instance-termination event matches a `source` of `aws.ec2`, a `detail-type` of `EC2 Instance State-change Notification`, and a `state` of `terminated`.

COMMON PITFALL

CloudWatch alarms do not act simply because a metric is *in* a breaching state at one moment. The state must have **changed and been maintained** for the configured number of evaluation periods before the alarm invokes its action — a subtle but frequently tested distinction between a momentary spike and a sustained problem.

Monitoring also manages cost. **AWS Cost Explorer** visualizes and manages costs and usage at daily or monthly granularity, with up to 13 months of data to reveal spending patterns. **AWS Budgets** sets custom budgets that alert you when costs or usage exceed — or are forecasted to exceed — a budgeted amount. The **AWS Cost and Usage Report**

contains the most comprehensive set of cost and usage data available, including metadata about services, pricing, and reservations.

10.2 Scaling your compute resources

When designing an application, you must ensure it runs optimally under varying conditions without manual intervention to recover from failures. Many modern applications handle petabytes of data, require close to 100 percent uptime, and deliver sub-second response times. To meet these demands, enterprises adopt **reactive architectures** — applications that are elastic, responsive, resilient, and message-driven.

CHARACTERISTICS OF A REACTIVE APPLICATION

- **Elastic** — scales resources dynamically, adding or removing capacity to avoid downtime and traffic bottlenecks.
- **Responsive** — responds in a timely manner with the lowest possible latency, even under varying workloads.
- **Resilient** — stays responsive in the face of failure, recovering when stressed by load, attacks, or component failure.
- **Message-driven** — relies on asynchronous message-passing for loose coupling, isolation, and location transparency (accessing data without knowing its location).

Elasticity is the ability of infrastructure to expand and contract as capacity requirements change — acquiring resources when needed and releasing them when not. **Scaling** is the ability to increase or decrease compute capacity, and it is the technique used to achieve elasticity. **Vertical scaling** changes the specifications of an individual resource: with EC2 you can stop an instance and resize it to a type with more RAM, CPU, I/O, or networking, and AWS transfers the data for you — but it is hardware-bound and can reach a limit. **Horizontal scaling** adds or removes resources available to the application (*scale out* adds, *scale in* ends), automatically transferring applications and data to the added resources.

VERTICAL (SCALE UP / DOWN)	HORIZONTAL (SCALE OUT / IN)
resize one resource	*add or remove resources*
▪ Resize one resource (more RAM, CPU, I/O)	▪ Add = **scale out**; remove = **scale in**
▪ Data transfer → usually **downtime**	▪ Good for internet-scale applications
▪ Hardware-bound — reaches a limit	▪ Uses the elasticity of cloud computing
▪ Not always cost-efficient or available	▪ Apps and data move to added resources

In the cloud, scaling can be automatic. **Amazon EC2 Auto Scaling** groups EC2 instances into an **Auto Scaling group** that can span Availability Zones. The group automatically adds or removes instances according to user-defined scaling policies, scheduled actions, and Elastic Load Balancing health checks; new instances are configured by a **launch template**. Auto Scaling integrates with ELB — registering new instances so traffic is distributed to them and receiving notifications of unhealthy instances to remove — and it balances instances evenly across Availability Zones. It is available free of charge.

BENEFITS OF EC2 AUTO SCALING

- **Better fault tolerance** — detects unhealthy instances, terminates them, launches replacements, and can span multiple AZs so another zone compensates when one fails.
- **Better availability** — helps ensure the application always has the right capacity for current traffic demand.
- **Better cost management** — increases and decreases capacity dynamically, so you launch instances only when needed and terminate them when they are not.

Group size is controlled by three capacity settings. **Minimum** is the smallest number of instances needed to run the application, **maximum** is the largest permitted, and **desired** is the optimal number under normal circumstances — adjusted as demand or the environment changes. You can scale manually (specifying a new maximum, minimum, or desired value) or automatically, but you can never exceed the maximum or drop below the minimum. Scaling begins with an event, or **scaling action**, that tells the group to launch or end instances. The launch template specifies the instance configuration (instance type, AMI ID) and can also specify what percentage of desired capacity is fulfilled with On-Demand, Reserved, and Spot Instances; Auto Scaling then provisions the lowest-price combination that meets desired capacity. A launch template can be versioned.

KEY TERMS

Desired capacity
The optimal number of instances to run the application under normal circumstances, between the minimum and maximum

Launch template
The versioned definition of instance configuration (instance type, AMI, purchase-option mix) used when the group launches instances

Scaling action
The event that instructs an Auto Scaling group to launch or end instances

WORKED EXAMPLE – DESIRED 2, MAXIMUM 3, MINIMUM 1

The group launches enough instances to meet its desired capacity of two (instances 1 and 2). If the load balancer reports an unhealthy instance, the group ends it and replaces it with a new instance. When a tracked metric from a service such as CloudWatch or Amazon SQS signals rising load, a scaling policy **scales out** by adding instance 3, reaching the maximum of three — and the group ensures maximum is not exceeded even if another scale-out notification arrives. Later, a scheduled scaling policy **scales in** during off-peak hours, ending instances 1 and 2 to reach the minimum of one instance.

A new Auto Scaling group has no scaling mechanisms; you add them. **Scheduled actions** scale as a function of date and time, ideal when you know exactly when load will change (for example, weekly traffic that rises Wednesday, peaks Thursday, and falls Friday). **Dynamic** scaling responds to changing conditions when you do not know when they will change. Its types are **target tracking** (select a metric and a target value — like a thermostat — and Auto Scaling creates and manages the alarms and calculates the adjustment), **step scaling** (adjust to match the size of the alarm breach), and **simple scaling** (wait for scaling to finish, ignoring subsequent alarms). **Predictive scaling** analyzes historical load with machine learning to forecast future capacity and proactively scale ahead of demand.

TABLE 10.1 Auto Scaling mechanisms and when to use them

MECHANISM	SCALES BASED ON	BEST SUITED FOR
Scheduled action	A date and time	Predictable workloads with known timing
Dynamic — target tracking	A target value for a specific metric	Keeping a metric steady (for example, CPU near 50%)
Dynamic — step	The size of the CloudWatch alarm breach	Moderately spiky load needing proportional response
Dynamic — simple	A CloudWatch alarm threshold	Simple adjustments; waits before responding again
Predictive	Forecast from historical patterns (ML)	Cyclical traffic, recurring on/off patterns, slow-to-initialize apps

TABLE 10.2 Step scaling policy example – adjustment by breach size

METRIC VALUE	CAPACITY CHANGE
Less than 30%	Remove 30% of current capacity
30–39%	Remove 10% of current capacity
40–60%	No change
61–70%	Add 10% of current capacity
71–100%	Add 30% of current capacity

Each step adjustment specifies a lower bound, an upper bound, and the amount to scale, and step scaling continues to respond to additional alarms while a scaling activity is in progress. Predictive and dynamic scaling can be used **together** to scale optimally. To scale more than EC2 instances, AWS offers two broader services. **AWS Auto Scaling** uses a **scaling plan** to configure scaling for multiple related resources at once — Amazon Aurora read replicas, EC2 Auto Scaling groups, Amazon ECS task counts, and DynamoDB throughput. **AWS Application Auto Scaling** scales individual resources beyond EC2 with target tracking, step, or scheduled policies — including AWS Lambda, Amazon SageMaker, and Amazon ElastiCache for Redis.

> **COMMON PITFALL**
>
> Launch configurations are legacy. Amazon EC2 Auto Scaling no longer supports launch configurations for instance types dated after December 2022. **Launch templates** are the preferred mechanism — they add versioning and support the full range of current instance features.

10.3 Scaling your databases

Like compute, AWS database services can scale vertically, horizontally, or both, but *how* differs for each service — some offer manual and automatic scaling, others only automatic scaling for certain components. Scaling a database is never truly instantaneous, although it can appear so because most of the work happens as **background processes**.

Amazon Aurora scales a **cluster**. Vertically, you change the Aurora DB instance class or size to resize compute and memory. Horizontally, **Aurora Auto Scaling** dynamically adjusts the number of Aurora Replicas (reader DB instances) — up to 15 — based on a scaling policy that defines minimum and maximum replicas and reacts to CloudWatch

metrics. Because changing the instance class can take **15 minutes or more**, you can minimize downtime by adding a replica of the target size, letting it synchronize, and then failing over to it. Aurora storage scaling in the cluster volume is managed automatically by the service.

Amazon Aurora Serverless is an on-demand, automatically scaling configuration for Aurora clusters that starts up, shuts down, and scales capacity based on demand without managing instances — ideal for infrequent, intermittent, or unpredictable workloads. You do not choose an instance class; instead you set minimum and maximum **Aurora Capacity Units (ACUs)**, where each ACU is roughly 2 GiB of memory with corresponding CPU and networking. Aurora Serverless scales within those limits, manages storage automatically, and lets you **start and stop** a cluster for intermittent workloads. You pay per second for the capacity used, and you can migrate between standard and serverless configurations.

Amazon RDS scales a **database**. Vertically, you change the DB instance class or size (the database is briefly unavailable during the change), and you can separately modify allocated storage or change the storage type — for example, from General Purpose SSD to Provisioned IOPS SSD — or use **RDS storage auto scaling**. On a **Multi-AZ** deployment, vertical scaling causes minimal downtime because the standby is upgraded first and then a failover occurs; a **Single-AZ** instance is unavailable during the operation. Horizontally, RDS uses built-in engine replication (MariaDB, MySQL, Oracle, PostgreSQL) to create **read replicas** — asynchronously updated, read-only copies that offload read queries and can serve reporting or disaster recovery, and can be promoted to primary.

Amazon DynamoDB scales a **table** by throughput. In **on-demand mode**, DynamoDB adapts rapidly to actual reads and writes — good for spiky or intermittent workloads. In **provisioned mode**, you set read capacity units (RCUs) and write capacity units (WCUs); by default it activates **auto scaling** through Application Auto Scaling to adjust provisioned throughput in response to traffic, increasing capacity to avoid throttling and decreasing it when load drops. Auto scaling also supports **global secondary indexes**, each of which has its own provisioned throughput separate from the base table. DynamoDB storage scaling is managed automatically by the service.

TABLE 10.3 Comparing AWS database scaling mechanisms

ASPECT	AURORA	AURORA SERVERLESS	AMAZON RDS	DYNAMODB
Scope	Cluster	Cluster	Database	Table
Vertical	Instance class or size	Cluster ACU throughput limits; Readers and Writers auto-scaled within limits	Instance class/size; storage auto scaling	N/A
Horizontal	Aurora Auto Scaling for read replicas	Multi-AZ reader replicas	Read replica databases	On-demand mode; provisioned mode with Application Auto Scaling; GSIs
Managed by service	Storage scaling	Storage scaling	N/A	Storage scaling

COMMON PITFALL

A read replica is not a high-availability failover mechanism. It is **read-only** and updated **asynchronously**; you can promote it to primary if the source becomes unavailable, but it is *not* a replacement for the automatic failover that **Multi-AZ** provides through a synchronously replicated standby.

10.4 Using load balancers to create highly available environments

When you build production-scale solutions, avoid **single points of failure** — one database and one application instance means the application fails if either fails. Plan for instance failure, Availability Zone failure, and Regional failure. A **highly available system** can withstand some degradation while remaining available, minimizing downtime and requiring minimal human intervention to return to normal. High availability enables the resiliency of a reactive architecture: the workload recovers from failure, or rolls over to a secondary source within an acceptable amount of degraded-performance time.

TABLE 10.4 Availability targets – the nines

UPTIME	MAX DOWNTIME PER YEAR	EQUIVALENT PER DAY
90%	36.5 days	2.4 hours
99%	3.65 days	14 minutes
99.9%	8.76 hours	86 seconds
99.99%	52.6 minutes	8.6 seconds
99.999%	5.25 minutes	0.86 seconds

Elastic Load Balancing (ELB) solves the single-point-of-failure problem for services such as EC2 instances and containers. It automatically distributes incoming traffic across multiple targets in one or more Availability Zones, can be external-facing (public traffic) or internal-facing (private traffic), monitors the **health** of registered targets, and routes traffic to only healthy ones. To check availability, the load balancer periodically pings, connects, or sends requests to targets; each target must respond with an **HTTP 200** status code to be considered healthy. If a target fails a health check, a notification is sent to the Auto Scaling group to replace it. ELB scales automatically as incoming traffic changes and gives each load balancer a default DNS name.

TABLE 10.5 The four ELB types by OSI layer

TYPE	OSI LAYER	DESIGNED FOR
Application Load Balancer	7 — application	HTTP/HTTPS and content-based routing to EC2, containers, IPs, and Lambda; supports Automatic Target Weights
Network Load Balancer	4 — transport	TCP/UDP/TLS and TLS offloading, static IPs, millions of requests per second at ultra-low latency
Gateway Load Balancer	3 — network	Deploy and scale third-party virtual appliances (firewalls, IDS/IPS) using the GENEVE protocol
Classic Load Balancer	3 & 7	Previous-generation EC2-Classic; AWS recommends ALB or NLB when possible

A load balancer is built from listeners, target groups, and health checks. A **listener** checks for connection requests using a protocol and port you configure; its **rules** each have a priority, one or more conditions, and one or more actions, and you must define a default rule. HTTPS and TLS listeners require TLS server certificates, which you provision with **AWS Certificate Manager (ACM)**; the load balancer uses a server certificate to terminate the frontend connection and decrypt requests before sending them to targets. A **target group** routes requests to registered targets by protocol and port, and each target group has its own health-check settings; the load balancer continually monitors registered targets in enabled AZs and routes only to healthy ones.

KEY TERMS

Listener
A process that checks for connection requests on a configured protocol and port, applying rules (priority, conditions, actions) with a required default rule

Target group
A group that routes requests to registered targets (EC2, containers, IPs, Lambda) by protocol and port, with its own health-check settings

Health check
A periodic ping, connection, or request that determines target availability; an HTTP 200 response marks a target healthy

AWS Certificate Manager (ACM)
Service to provision, manage, and deploy SSL/TLS certificates for services such as load balancers

> WORKED EXAMPLE – HA WITH AN ALB AND RDS MULTI-AZ
>
> An external-facing **Application Load Balancer** receives public HTTP/HTTPS traffic (routed through Route 53) and distributes it to a web-tier target group of EC2 instances in an Auto Scaling group across AZ1 and AZ2, using round robin by default (or least outstanding requests). If one instance fails its health check, the ALB stops sending it traffic and notifies the Auto Scaling group to replace it; if a whole AZ fails, the ALB sends traffic only to healthy instances in the other AZ. An internal-facing second ALB follows the same pattern for the app tier. The data tier uses **RDS Multi-AZ**: the primary in AZ1 synchronously replicates to a secondary in AZ2, and on failure the secondary is promoted to primary. The entire architecture can be duplicated to another Region for Regional resilience.

A **Network Load Balancer** (layer 4) can expose a centralized service and handle millions of requests per second, supporting client connections over VPC peering, VPC endpoints, AWS managed VPN, AWS Direct Connect, and third-party VPNs. You can register a single Application Load Balancer as an NLB target to combine layer-7 request routing with NLB features such as static IP addresses and endpoint services — useful for applications needing one endpoint for multiple protocols.

A **Gateway Load Balancer** (layer 3) is a security solution that scans incoming and outgoing traffic with virtual appliances. It listens for all IP packets across all ports, maintains flow stickiness to a specific appliance using 5-tuple (TCP/UDP) or 3-tuple (non-TCP/UDP), and exchanges traffic with its appliance instances using the **GENEVE** protocol. Traffic entering a VPC is routed to a Gateway Load Balancer endpoint, sent to the Gateway Load Balancer and its EC2 security appliance for inspection, then returned to the endpoint and forwarded to the application instance — unlike an NLB, which exposes services and distributes network traffic across servers.

10.5 Using Route 53 to create highly available environments

The **Domain Name System (DNS)** is a globally distributed service that translates human-readable names such as www.myweb.com into the numeric IP addresses computers use to connect. Requests for names are called **queries**. Clients usually do not query authoritative DNS servers directly; they connect to a **resolver** (recursive DNS service), typically run by their ISP, which owns no records but fetches DNS information on your behalf. If the resolver has the answer cached, it responds immediately; otherwise it passes the query up the hierarchy.

WORKED EXAMPLE – RESOLVING WWW.MYWEB.COM

A user enters www.myweb.com; the request goes to the ISP's DNS **resolver**. The resolver forwards it to a DNS **root name server**, which returns the location of the .com **top-level domain (TLD)** name server. The resolver then asks the .com TLD server, which responds with the names of the authoritative name servers for the domain. The resolver forwards the request to one of those **authoritative name servers**, which looks in the myweb.com **hosted zone**, finds the www record, and returns the IP address (for example, 192.0.3.4). The resolver returns that IP to the browser and **caches** it for a time you specify so the next lookup is faster; the browser then requests the page directly from that IP.

Amazon Route 53 is a highly available, scalable DNS web service that performs three main functions in any combination: **domain registration**, **DNS routing**, and **health checking**. You can register and manage domain names, and Route 53 configures DNS settings automatically. A **hosted zone** is a container for records describing how to route traffic for a domain and all its subdomains, and it has the same name as the domain. Route 53 provides authoritative name servers, connects requests to AWS resources (EC2, ELB, S3) or resources outside AWS, and performs **health checks** — endpoint checks, calculated checks that watch other checks, or CloudWatch alarm checks. Route 53 Application Recovery Controller adds **zonal autoshift**, which automatically shifts traffic away from an Availability Zone AWS identifies as potentially failing, and **routing controls** (on/off switches) that let you coordinate failover across AZs or Regions.

Route 53 supports multiple **routing policies** that determine how it responds to queries. Choosing a policy is a key high-availability decision, and several can be combined with DNS failover to build low-latency, fault-tolerant architectures.

TABLE 10.6 Route 53 routing policies

POLICY	WHAT IT DOES
Simple	Standard DNS records with no special routing; typically routes to a single resource
Weighted	Sends proportions of traffic to multiple resources by assigned weight; useful for load balancing and testing new versions
Latency	Serves requests from the AWS Region that gives the user the lowest latency
Failover	Routes to a primary resource while healthy and to a secondary when the primary is unhealthy
Geoproximity	Routes by the geographic location of users and resources, with an optional bias to expand or shrink a region
Geolocation	Routes by the location the DNS query originates from; supports content localization and distribution restrictions
Multivalue answer	Returns multiple healthy IP addresses; improves availability but is not a substitute for a load balancer
IP-based	Routes using your own user-IP-to-endpoint mappings to optimize performance or reduce network cost

To route traffic you create a **public** hosted zone (internet traffic) or a **private** hosted zone (VPC traffic), then add records that define where to route each name. When creating a record you choose a routing policy, an optional failover destination, and a record type — commonly **A** (IPv4 address), **AAAA** (IPv6 address), **CNAME** (maps one name to another domain), **MX** (mail servers and priority), and **NS** (name servers for the zone). **Route 53 Resolver** responds recursively to queries from AWS resources for public records, VPC-specific DNS names, and private hosted zones, and is available by default in all VPCs.

For multi-Region high availability, configure **DNS failover** so Route 53 routes from an unhealthy resource to a healthy one. Use an **active-passive** configuration when a primary resource should serve most of the time and a secondary stands by: while the primary is healthy Route 53 answers with the primary record, and when it becomes unhealthy Route 53 answers with the secondary record. Route 53 can check health in **simple** configurations (a group of same-name, same-type records) and in **complex** configurations (a tree of records combining, for example, latency alias records with weighted records that route to specific instance types).

10.6 Applying AWS Well-Architected Framework principles to highly available systems

The **AWS Well-Architected Framework** has six pillars, each with best practices and questions to consider when architecting solutions. This section highlights practices from the **reliability** and **performance efficiency** pillars most relevant to high availability. A system that is *available* can deliver its designed functionality at a given point in time; a *highly available* system can withstand some degradation while still remaining available.

Use fault isolation to protect your workload. Fault isolation boundaries limit the effect of a failure to a small number of components so problems do not affect the whole system. Two best practices apply. **Deploy the workload to multiple locations** — distribute data and resources across multiple Availability Zones or, where necessary, across AWS Regions, avoiding single points of failure in physical infrastructure and ensuring redundant components operate independently and (across Regions) autonomously. **Automate recovery for components constrained to a single location** — where multi-location deployment is not possible, automate recreating infrastructure, redeploying applications, and recreating data; use automatic scaling, automatic recovery for EC2, or self-healing automation. Services such as Amazon EC2 Auto Scaling and Amazon RDS Multi-AZ support these practices.

Design your workload to withstand component failures. Always distribute load across resources, Availability Zones, or Regions so that traffic can shift to remaining healthy resources. **Fail over to healthy resources** — for location impairments (an AZ or Region), have systems in place to fail over to healthy resources in unimpaired locations, and monitor that the failover resource is healthy and that business processes recover. **Send notifications when events impact availability** — because automated healing can obscure underlying problems, send alerts immediately to operations teams when thresholds such as error rate, latency, or other KPIs are breached, even if the issue self-resolved. Route 53, CloudWatch, and Amazon EC2 Auto Scaling support these practices.

> **WELL-ARCHITECTED HA BEST PRACTICES COVERED IN THIS MODULE**
>
> - Deploy the workload to multiple locations.
> - Automate recovery for components constrained to a single location.
> - Fail over to healthy resources.
> - Send notifications when events impact availability.
> - Scale your compute resources dynamically.

Scale your compute resources dynamically (performance efficiency). AWS provides several mechanisms to match the supply of resources to demand, and deployments must handle both scale-up and scale-down events: a **target-tracking** approach monitors a scaling metric and adjusts capacity automatically; **predictive scaling** scales in anticipation of daily and weekly trends; a **schedule-based** approach follows a predictable load schedule; and **service scaling** relies on services (such as serverless) that scale by design. ELB, CloudWatch, and Amazon EC2 Auto Scaling support this practice.

SAMPLE EXAM QUESTION – FIND THE SINGLE POINT OF FAILURE

A web application lets customers upload orders to an S3 bucket. The resulting S3 events invoke a Lambda function that inserts a message into an SQS queue. A single EC2 instance reads messages from the queue, processes them, and stores them in a DynamoDB table partitioned by order ID. Traffic will increase tenfold next month. Which component is MOST likely to need re-architecting to scale? Isolate the key phrases: **a single EC2 instance** and **able to scale**. The Lambda function, SQS queue, and DynamoDB table are all managed services that scale automatically or can be configured to scale. The answer is the **EC2 instance** — a single instance does not scale and is a single point of failure. The better design places EC2 instances in an **Auto Scaling group across two Availability Zones**, all reading from the queue.

Module summary

- Monitoring is the foundation of reactive architecture: CloudWatch collects metrics and logs across Regions, calculates statistics, and displays them on dashboards that can span multiple Regions.
- A CloudWatch alarm watches one metric and, only on a sustained state change across its evaluation periods, sends its action to an SNS topic or an Auto Scaling policy; EventBridge routes events to as many as five targets per rule via event patterns or schedules.
- Cost monitoring uses AWS Cost Explorer (up to 13 months, daily/monthly), AWS Budgets (alerts on actual or forecasted overage), and the AWS Cost and Usage Report (most comprehensive data).
- Reactive applications are elastic, responsive, resilient, and message-driven; elasticity is achieved through scaling — vertical (resize one resource, with downtime) or horizontal (add/remove resources, scale out/in).
- An EC2 Auto Scaling group spans Availability Zones, is bounded by minimum/maximum/desired capacity, launches instances from a launch template, integrates with ELB for registration and health checks, and is free beyond the instances.
- Scaling mechanisms are scheduled actions (predictable time), dynamic policies (target tracking, step, simple), and predictive scaling (ML on history); AWS Auto Scaling and Application Auto Scaling extend scaling beyond EC2 to Aurora, ECS, DynamoDB, Lambda, and more.
- Databases scale differently: Aurora (instance class plus read replicas), Aurora Serverless (min/max ACUs), RDS (instance class/storage plus read replicas, Multi-AZ for HA), DynamoDB (on-demand or provisioned with Application Auto Scaling); storage scaling is service-managed for all but RDS.
- Highly available systems avoid single points of failure and are measured in nines of uptime; ELB distributes traffic to healthy targets in multiple AZs via listeners, target groups, and health checks (HTTP 200).
- The four ELB types map to OSI layers — ALB (7), NLB (4), GWLB (3), CLB (3 & 7) — and combine with RDS Multi-AZ to eliminate single points of failure within a Region.
- Route 53 provides DNS registration, routing, and health checking with eight routing policies and active-passive failover for multi-Region HA; the Well-Architected reliability and performance-efficiency pillars capture the pattern — multiple locations, fail over to healthy resources, notify, and scale dynamically.

Review questions

1. What is the difference between a metric's standard and high resolution, and what does that mean for alarms?

Answer: Standard resolution stores data points at 1-minute granularity; high resolution at 1-second granularity. A high-resolution metric can use a high-resolution alarm with a 10- or 30-second period (at a higher charge), whereas standard-resolution metrics use regular alarms with periods that are multiples of 60 seconds.

2. Why does a distributed application need centralized logging, and how does CloudWatch provide it?

Answer: Passing messages between many services makes failures hard to reproduce and debug. CloudWatch Logs consolidates logs from every service, EC2, CloudTrail, Route 53, VPC, and more into a single, time-ordered, searchable flow, from which metrics and alarms can be generated.

3. Contrast vertical and horizontal scaling, and say which supports high availability.

Answer: Vertical scaling replaces a resource with a larger or smaller one — hardware-bound and usually requiring downtime. Horizontal scaling adds or removes resources (scale out/in), moving data automatically and enabling internet-scale, highly available designs across Availability Zones.

4. A target tracking policy targets 50% average CPU. During a stress test that pushes CPU higher, what does Auto Scaling do and why?

Answer: It launches more instances. The tracking metric exceeded the 50% target, so the group scales out to bring average CPU back down toward the target; CloudWatch alarms that the policy manages drive the adjustment.

5. How do Aurora, Amazon RDS, and DynamoDB each scale horizontally?

Answer: Aurora uses Aurora Auto Scaling to add or remove read replicas (up to 15); RDS adds asynchronous read replicas; DynamoDB uses on-demand mode or provisioned mode with Application Auto Scaling on RCUs/WCUs, plus separately scaled global secondary indexes.

6. In an ALB plus RDS Multi-AZ design, what happens when one EC2 instance fails, and when the primary database's AZ fails?

Answer: The ALB stops sending traffic to the failed instance and notifies the Auto Scaling group to terminate and replace it. If the primary database or its AZ fails, RDS promotes the synchronously replicated secondary in the other AZ to primary.

7. Name the three functions of Route 53 and the routing policy used for active-passive multi-Region high availability.

Answer: Domain registration, DNS routing, and health checking. Active-passive HA uses failover routing — Route 53 answers with the primary record while it is healthy and switches to the secondary when it is not.

8. State two Well-Architected best practices for high availability from this module and a service that supports each.

Answer: Deploy the workload to multiple locations (EC2 Auto Scaling, RDS Multi-AZ); fail over to healthy resources and send notifications when events impact availability (Route 53, CloudWatch); scale compute dynamically (ELB, CloudWatch, EC2 Auto Scaling).