MODULE 5 · STUDY NOTES

# Networking and Content Delivery

*Networking fundamentals, Amazon VPC and its building blocks, VPC connectivity and security, Route 53 DNS, and the CloudFront CDN*

Everything in AWS talks over a network, and this module builds that network from the ground up. It starts with the raw materials — IP addresses, CIDR notation, the OSI model — then constructs an Amazon VPC out of them: subnets, route tables, gateways, Elastic IPs, and network interfaces. From there it wires the VPC to the outside world (internet gateways, NAT, peering, VPN, Direct Connect, endpoints, Transit Gateway), wraps it in two firewall layers (security groups and network ACLs), and finishes with the two services that carry traffic to users at global scale: Route 53 for DNS and CloudFront for content delivery.

LEARNING OBJECTIVES

- Recognize the basics of networking
- Describe virtual networking in the cloud with Amazon VPC
- Label a network diagram
- Design a basic VPC architecture
- Indicate the steps to build a VPC
- Identify security groups
- Create your own VPC and add additional components to it to produce a customized network
- Identify the fundamentals of Amazon Route 53
- Recognize the benefits of Amazon CloudFront

## 5.1 Networking basics

A **computer network** is two or more client machines connected to share resources, joined by a networking device such as a **router or switch** that enables communication between them. A network can be logically partitioned into **subnets**. Each machine on a network carries a unique **IP address** — a numerical label written in decimal that machines convert to binary. An **IPv4** address like `192.0.2.0` is **32 bits**: four dot-separated numbers, each representing 8 bits, so each ranges 0–255. An **IPv6** address is **128 bits** — eight colon-separated groups of four hexadecimal characters (each group 16 bits, `0`–`FFFF`) — and exists to accommodate far more devices.

**Classless Inter-Domain Routing (CIDR)** is the common way to describe a network: an IP address (the first address of the network), a slash, and a number saying how many leading bits of the routing prefix are **fixed** as the network identifier. The remaining bits are free to change, so a CIDR block names a run of consecutive addresses.

WORKED EXAMPLE – READING A CIDR BLOCK

`192.0.2.0/24`: the first 24 bits are fixed, the last 8 are flexible → $2^8$ = **256** addresses, ranging 192.0.2.0–192.0.2.255 (only the fourth number changes). `192.0.2.0/16`: 16 bits fixed, 16 flexible → $2^{16}$ = **65,536** addresses, 192.0.0.0–192.0.255.255 (third and fourth numbers change). Two special cases: **/32** fixes every bit — a single IP address, handy for a firewall rule granting one specific host access — and **`0.0.0.0/0`** fixes nothing, which means *the entire internet*.

The **Open Systems Interconnection (OSI) model** is a seven-layer conceptual model explaining how data travels over a network, showing the common protocols and addresses used at each layer — and it applies just as well to communication inside a VPC. The anchors to remember: **hubs and switches work at layer 2** (data link); **routers work at layer 3** (network).

**TABLE 5.1** The OSI model, bottom to top

| LAYER | NAME | IN THIS MODULE |
|---|---|---|
| **7** | Application | Protocols applications speak (e.g. HTTP for web traffic) |
| **6** | Presentation | Data formats and encryption |
| **5** | Session | Connections between hosts |
| **4** | Transport | End-to-end delivery (TCP/UDP) |
| **3** | Network | IP addressing and routing — **routers live here** |
| **2** | Data link | Local delivery between adjacent nodes — **hubs and switches live here** |
| **1** | Physical | Cables, signals, bits on the wire |

## 5.2 Amazon VPC and its building blocks

**Amazon VPC** lets you provision a **logically isolated section of the AWS Cloud** — a virtual private cloud — and launch AWS resources into a virtual network **you define**. You control the virtual networking resources: choice of IP address range, creation of subnets, and configuration of route tables and network gateways, with both IPv4 and IPv6 supported. You can customize the layout — a public subnet for web servers with internet access, private subnets for backend databases and application servers without it — and apply **multiple layers of security** (security groups and network ACLs) to control access to EC2 instances in each subnet.

> **VPCS AND SUBNETS – SCOPING RULES**
>
> - A VPC is logically isolated from other VPCs and **dedicated to your AWS account**.
> - A VPC belongs to **a single Region** and can span **multiple Availability Zones**.
> - A subnet is a range of IP addresses dividing the VPC; each subnet belongs to **a single AZ** — create subnets in different AZs for high availability.
> - Subnets are classified **public** (direct internet access) or **private** (none).

**IP addressing.** When you create a VPC you assign it an IPv4 CIDR block of private addresses — and **you cannot change that range afterward**, so choose carefully. The block can be as large as **/16** (65,536 addresses) or as small as **/28** (16 addresses). An IPv6 block can optionally be associated (with a different size limit). A subnet's CIDR can equal the VPC's (one big subnet) or be a subset of it; if you create multiple subnets, their **CIDR blocks cannot overlap** — no duplicate IP addresses in the same VPC.

**Reserved addresses.** In every subnet CIDR block, AWS reserves **five addresses** that you cannot use: the **network address**, the **VPC local router** (internal communications), **DNS resolution**, one held for **future use**, and the **network broadcast address**. So a /24 subnet (256 total) offers 251 usable addresses — the deck's example VPC of `10.0.0.0/16` split into four equal subnets yields only **251 usable addresses per subnet**.

**TABLE 5.2** Getting a public IP address

| TYPE | HOW IT WORKS |
|---|---|
| **Automatic public IPv4** | Enabled via the subnet's **auto-assign public IP** setting when the instance is created (every instance always gets a private IP from the VPC range automatically) |
| **Elastic IP address** | A **static public IPv4** associated with your **account**, allocatable and remappable to any instance or network interface anytime — remapping masks an instance failure fast. Additional costs may apply, so **release** unused EIPs |

An **elastic network interface** is a virtual network interface you can attach to an instance, detach, and attach to another instance — its attributes follow it, and network traffic redirects to the new instance. Every instance has a default (**primary**) network interface holding a private IPv4 address from the VPC range; the primary interface **cannot be detached**, and how many extra interfaces an instance supports varies by instance type. Tip from the deck: associating an Elastic IP with a network interface rather than directly with the instance lets you move all its attributes to another instance in a single step.

A **route table** contains rules (**routes**) directing network traffic from a subnet. Each route names a **destination** (the CIDR block traffic is bound for) and a **target** (what carries it there). Every route table contains a built-in **local route** for communication within the VPC, which **cannot be deleted**; you customize by adding routes. Every subnet must be associated with **exactly one** route table at a time — though one route table can serve many subnets — and any subnet not explicitly associated uses the VPC's **main route table**.

> COMMON PITFALL
>
> Two fixed-at-creation facts the knowledge check likes: the VPC CIDR block **cannot be changed** after creation, and the **local route** in a route table cannot be deleted. And remember the direction of the one-to-many rule: a subnet has *one* route table; a route table can serve *many* subnets.

## 5.3 VPC networking: connecting to everything

An **internet gateway** is a scalable, redundant, highly available VPC component that allows communication between instances in your VPC and the internet. It serves **two purposes**: it is the **target in route tables** for internet-routable traffic, and it performs **network address translation** for instances with public IPv4 addresses. To make a subnet public: attach an internet gateway to the VPC, then add a route sending non-local traffic (`0.0.0.0/0`) through it.

A **NAT gateway** enables instances in a *private* subnet to connect out to the internet or other AWS services while **preventing the internet from initiating connections to them**. Creating one requires two decisions: it must be placed in a **public subnet**, and it needs an **Elastic IP address**. Then update the private subnets' route table(s) to point internet-bound traffic at the NAT gateway. You could self-manage a **NAT instance** in a public subnet instead, but AWS recommends the NAT gateway: it is a managed service with better availability, higher bandwidth, and less administrative effort.

**VPC sharing** lets the account that owns a VPC (**owner**) share subnets with other accounts (**participants**) in the same organization in **AWS Organizations**. Participants can view, create, modify, and delete their own application resources (EC2 instances, RDS databases, Redshift clusters, Lambda functions) in shared subnets — but cannot touch resources belonging to other participants or the owner. Benefits: **separation of duties** (centrally controlled VPC structure, routing, IP allocation), **ownership** (application owners keep their resources, accounts, and security groups), cross-account **security-group referencing**, **efficiencies** (higher subnet density, efficient VPN/Direct Connect use), avoidance

of hard limits, and **optimized costs** through reuse of NAT gateways, VPC endpoints, and intra-AZ traffic. The net effect: fewer, larger, centrally managed VPCs — accounts decoupled from networks.

A **VPC peering connection** routes traffic **privately** between two VPCs, so instances communicate as if on the same network — between your own VPCs, across AWS accounts, or across Regions. Both sides add routes: in VPC A's table, destination = VPC B's IP range, target = the peering resource ID (and vice versa). Three restrictions: **IP address ranges cannot overlap**, **transitive peering is not supported** (A↔B plus A↔C does not connect B and C — that link must be created explicitly), and there can be only **one peering resource between the same two VPCs**.

**AWS Site-to-Site VPN** connects a VPC to a remote network — by default, VPC instances cannot talk to remote networks at all. The sequence: create a **VPN gateway** (virtual private gateway) and attach it to the VPC; define the **customer gateway** (an AWS resource carrying information about your physical VPN device, not the device itself); create a custom route table pointing corporate-bound traffic at the VPN gateway and update security group rules; establish the Site-to-Site VPN connection; configure routing to pass traffic through it.

**AWS Direct Connect (DX)** answers the performance problem of being far from your AWS Region: a **dedicated, private network connection** between your network and a DX location. It can reduce network costs, increase bandwidth throughput, and deliver a **more consistent network experience** than internet-based connections. DX uses open-standard **802.1q VLANs**.

A **VPC endpoint** is a virtual device that connects your VPC **privately** to supported AWS services — no internet gateway, NAT device, VPN, or Direct Connect required, no public IPs needed, and traffic **never leaves the Amazon network**. Two types: **interface endpoints** (powered by **AWS PrivateLink**) reach AWS services, partner-hosted endpoint services, and Marketplace services — billed hourly plus data processing; **gateway endpoints** support **Amazon S3 and DynamoDB** and carry **no additional charge**.

**AWS Transit Gateway** rescues you from mesh sprawl. Customers commonly run hundreds of VPCs across accounts and Regions, and every connectivity option above is strictly **point-to-point** — pairwise peering and per-VPC VPN attachments become operationally costly and hard to manage at scale. A transit gateway is a **hub**: you create and manage a **single connection** from it to each VPC, on-premises data center, or remote office, and it controls how traffic routes among all the connected **spokes**. Any new VPC connected to the hub is automatically reachable from every other connected network — dramatically simpler management as you grow.

**TABLE 5.3** VPC connectivity options – the section's key takeaway list

| OPTION | ONE-LINE JOB |
|---|---|
| **Internet gateway** | Connects your VPC to the internet |
| **NAT gateway** | Enables instances in a private subnet to connect out to the internet |
| **VPC endpoint** | Connects your VPC to supported AWS services privately |
| **VPC peering** | Connects your VPC to other VPCs |
| **VPC sharing** | Lets multiple accounts create resources in shared, centrally managed VPCs |
| **Site-to-Site VPN** | Connects your VPC to remote networks over VPN |
| **Direct Connect** | Connects your VPC to a remote network over a dedicated network connection |
| **Transit Gateway** | Hub-and-spoke alternative to point-to-point VPC peering |

The module's activity asks you to label these components on a network diagram, and a recorded demonstration shows the **VPC Wizard** creating a VPC with public and private subnets — the same tool the lab uses to implement your design.

## 5.4 VPC security: security groups and network ACLs

You control both incoming and outgoing traffic by building security into the VPC architecture itself — isolate subnets where possible, choose the right gateway or VPN for each need, and then apply the two firewall layers this section covers. A **security group** is a virtual firewall that controls inbound and outbound traffic **at the instance level** — each instance in a subnet can be assigned a different set of security groups. A brand-new custom security group has **no inbound rules** (so no inbound traffic is allowed until you add some) and one outbound rule **allowing all outbound traffic** (removable and replaceable with specific rules). Security groups are **stateful**: state is kept after a request is processed, so responses to an outbound request flow back in regardless of inbound rules, and responses to allowed inbound traffic flow out regardless of outbound rules. You can specify **allow rules only** — never deny — and **all rules are evaluated** before the decision to allow traffic.

A **network access control list (network ACL)** is an *optional* extra security layer acting as a firewall **at the subnet level**. Every subnet must be associated with a network ACL — with the **default network ACL** if you don't choose one — and while one nACL can cover many subnets, a subnet has only **one nACL at a time** (a new association replaces the old). A network ACL holds separate **inbound and outbound rules**, and each rule can **allow or deny**. The default nACL **allows all** inbound and outbound IPv4 traffic; a **custom** nACL **denies everything** until you add rules. Network ACLs are **stateless** — nothing about a request is remembered, so return traffic must be explicitly permitted. Rules are numbered and evaluated **in order from the lowest number** (maximum rule number 32,766); AWS recommends numbering in increments of 10 or 100 so you can insert rules later.

| SECURITY GROUP | NETWORK ACL |
|---|---|
| *instance-level firewall* | *subnet-level firewall* |
| ▪ Operates at the **instance** level | ▪ Operates at the **subnet** level |
| ▪ **Allow** rules only | ▪ **Allow and deny** rules |
| ▪ **Stateful** — return traffic automatically allowed | ▪ **Stateless** — return traffic needs explicit rules |
| ▪ **All rules evaluated** before deciding | ▪ Rules evaluated in **number order**, lowest first |
| ▪ Default: deny all inbound, allow all outbound | ▪ Default nACL allows all; custom denies all until rules added |

**COMMON PITFALL**

The stateful/stateless distinction is the classic exam discriminator. Stateful (security group): a reply to traffic you initiated is always let back through. Stateless (network ACL): the reply is brand-new traffic and must match an explicit inbound *and* outbound rule. If a web server behind a custom nACL times out on responses, the missing outbound rule is the usual culprit.

**WORKED SCENARIO – THE DECK'S "DESIGN A VPC" ACTIVITY**

*Requirements:* a small-business website on EC2 with a private backend database; web and database servers in separate subnets; network starts at 10.0.0.0 with 256 total IPv4 addresses per subnet; customers must always reach the web server; the database must reach the internet for patch updates; the architecture must be highly available with at least one custom firewall layer. *One solution:* a VPC (e.g. `10.0.0.0/16`) with a **public subnet** `10.0.0.0/24` (web server) and a **private subnet** `10.0.1.0/24` (database) — /24 = 256 total addresses each. Attach an **internet gateway** and route `0.0.0.0/0` → IGW in the public route table so customers reach the web server; put a **NAT gateway** (with an Elastic IP) in the public subnet and route the private subnet's internet-bound traffic to it for patching. For high availability, mirror both subnets into a **second Availability Zone**; for the firewall layer, add **security groups** (and optionally a custom network ACL) around the tiers.

**Lab 2 — Build Your VPC and Launch a Web Server** (~30 minutes) turns this into muscle memory: create a VPC, create additional subnets, create a VPC security group, and launch a web-server EC2 instance into the VPC using that security group.

## 5.5 Amazon Route 53

**Amazon Route 53** is a highly available and scalable **Domain Name System (DNS)** web service: it routes end users to internet applications by translating names like `www.example.com` into the numeric IP addresses (like `192.0.2.1`) that computers use to connect. It is fully compliant with both IPv4 and IPv6. Route 53 connects user requests to infrastructure **running in AWS** — EC2 instances, Elastic Load Balancing load balancers, S3 buckets — and equally to infrastructure **outside AWS**. Beyond resolution, it offers **DNS health checks** (route traffic to healthy endpoints, or independently monitor application health), **traffic flow** (a visual editor managing global routing, combining routing types with DNS failover for low-latency, fault-tolerant architectures), and **domain name registration** with automatic DNS configuration. The resolution pattern itself is simple: the DNS resolver checks with your domain in Route 53, gets the IP address, and returns it to the user.

**TABLE 5.4** Route 53 routing policies

| POLICY | WHAT IT DOES | TYPICAL USE |
|---|---|---|
| **Simple** | One record answers for the domain (round robin) | Single-server environments |
| **Weighted round robin** | Weights 0–255 set response frequency — weight 3 vs. 1 returns the heavier record 75% of the time | A/B testing a software change on a small slice of traffic |
| **Latency (LBR)** | Routes to the AWS endpoint with the fastest **measured** performance | Applications in multiple Regions |
| **Geolocation** | Routes by the **user's** location | Localized content/language; distribution rights; predictable load per locale |
| **Geoproximity** | Routes by the **resources'** location; can shift traffic between locations | Gradually rebalancing between sites |
| **Failover** | Active-passive: health checks detect an outage, users go to the backup | DNS failover to a standby site |
| **Multivalue answer** | Up to **eight healthy records**, selected at random | DNS-level availability boost — *not* a load-balancer substitute |

The showcase use case is **multi-Region deployment**: Route 53 automatically directs each user to the Elastic Load Balancing load balancer closest to them — **latency-based routing to the Region**, then **load balancing across Availability Zones** within it. For availability, Route 53 improves applications through backup-and-failover configurations, highly available multi-Region architectures, and **health checks**, each of which can monitor one of three things: the health of a specified resource (such as a web server), the status of **other health checks**, or the status of a **CloudWatch alarm**.

> **WORKED SCENARIO – DNS FAILOVER FOR A MULTI-TIERED WEB APP**
>
> Route 53 passes traffic to a load balancer, which distributes it to a fleet of EC2 instances backed by a database. Create **two CNAME records for** `www` with **failover routing**: the primary route policy points at the load balancer of the web application; the secondary points at a **static Amazon S3 website**. Route 53 health checks watch the primary — as long as it passes, all traffic goes to the application stack. If the web server goes down (or stops responding) or the database instance fails, failover triggers and users land on the static backup site instead of an error page.

## 5.6 Amazon CloudFront

Content delivery happens over networks too, and it inherits their central problem: **latency**. A browser request hops through many networks to reach the **origin server** — the machine storing the original, definitive versions of webpages, images, and media — and the number of hops and distance traveled directly affect performance, which also varies by geography. The fix is a **content delivery network (CDN)**: a **globally distributed system of caching servers** that caches copies of commonly requested **static content** (HTML, CSS, JavaScript, images) and serves a local copy from the nearest cache edge or **Point of Presence (PoP)**. CDNs also accelerate **dynamic** content that is unique to the requester and not cacheable — by maintaining secure connections close to the requester and fast routes back to the origin — improving application performance and scaling.

**Amazon CloudFront** is a fast, **global, and secure** CDN service that delivers data, videos, applications, and APIs to customers worldwide with low latency and high transfer speeds. Unlike traditional CDN solutions, there are no negotiated contracts, high prices, or minimum fees — it is **self-service** with **pay-as-you-go** pricing. Delivery runs over a global network of **edge locations** and **Regional edge caches**: users are routed to the edge location with the lowest latency, which serves popular content quickly. As objects lose popularity, edge locations evict them — and the **Regional edge caches** catch them: deployed globally between your origin server and the edge locations, they have **larger caches**, so objects stay cached longer and closer to viewers, reducing trips back to the origin.

> **CLOUDFRONT'S FIVE BENEFITS (A KEY-TAKEAWAY LIST)**
>
> - **Fast and global** — massively scaled, globally distributed network of edge locations and regional caches.
> - **Security at the edge** — network- and application-level protection; **AWS Shield Standard** built in at no extra cost; custom SSL certificates via **AWS Certificate Manager**, also free.
> - **Highly programmable** — **Lambda@Edge** runs custom code at AWS locations worldwide, moving application logic closer to users; integrates with DevOps tooling and CI/CD environments.
> - **Deeply integrated with AWS** — physical connections to the AWS Global Infrastructure; configure everything via APIs or the console.
> - **Cost-effective** — no minimum commitments; collapses simultaneous viewer requests for the same file into a single origin request; with AWS origins (S3, ELB) you pay storage only, **not** data transfer between those services and CloudFront.

**TABLE 5.5** CloudFront pricing – the four billed areas

| AREA | HOW YOU'RE CHARGED |
|---|---|
| **Data transfer out** | Per GB transferred from edge locations to the internet or to your origin; totaled per geographic region with tiered pricing (AWS-origin storage/compute billed separately) |
| **HTTP(S) requests** | Per request made to CloudFront for your content |
| **Invalidation requests** | Per **path** (a URL, or several via wildcard) removed from cache — first **1,000 paths per month free**, then $0.005 per path |
| **Dedicated IP custom SSL** | **$600/month** per custom SSL certificate on the Dedicated IP option, **prorated by the hour** — e.g. associated for 1 day in June = (1/30) × $600 = **$20** |

COMMON PITFALL

Don't invert the cache hierarchy: **edge locations** hold the *popular* content next to viewers; **Regional edge caches** hold the *less-popular* overflow, sitting between the origin and the edges with a larger cache. More content near viewers = fewer origin fetches.

## Module summary

- IPv4 addresses are 32-bit, IPv6 are 128-bit; CIDR `x.x.x.x/n` fixes the first *n* bits — /24 = 256 addresses, /16 = 65,536, /32 = one host, 0.0.0.0/0 = the internet.
- A VPC is a logically isolated section of the AWS Cloud in one Region spanning AZs; each subnet sits in one AZ; the VPC CIDR (/16 down to /28) is unchangeable after creation; AWS reserves 5 addresses per subnet.
- Route tables pair destinations with targets, carry an undeletable local route, and attach one-per-subnet (a table can serve many subnets).
- Public subnet = 0.0.0.0/0 route to an internet gateway; private instances reach out through a NAT gateway placed in a public subnet with an Elastic IP — outbound only.
- Connectivity menu: VPC peering (non-transitive, no overlap, one per pair), VPC sharing, Site-to-Site VPN, Direct Connect (dedicated line, 802.1q), interface/gateway endpoints (gateway = S3 + DynamoDB, free), and Transit Gateway's hub-and-spoke.
- Security groups: instance-level, stateful, allow-only, all rules evaluated. Network ACLs: subnet-level, stateless, allow + deny, numbered rules evaluated lowest first.
- Route 53 is highly available, scalable DNS (names → IPs) with seven routing policies, health checks, DNS failover, and domain registration; multi-Region + latency routing serves global audiences.
- CloudFront is a pay-as-you-go CDN over edge locations (popular content) and Regional edge caches (less-popular content); billed on data out, requests, invalidations, and dedicated-IP custom SSL.

## Review questions

**1. Why does AWS make you commit to a VPC CIDR block at creation time, and what are the size limits?**

*Answer:* The address range cannot be changed after the VPC is created, so it must be planned up front — between /16 (65,536 addresses) and /28 (16 addresses) for IPv4, with subnet blocks drawn from it that may not overlap.

**2. Your subnet is 10.0.0.0/24. How many instances can it actually hold, and why?**

*Answer:* 251. A /24 contains 256 addresses, but AWS reserves five in every subnet: the network address, the VPC local router, DNS, one for future use, and the broadcast address.

**3. What exactly makes a subnet 'public'?**

*Answer:* An internet gateway is attached to the VPC and the subnet's route table sends non-local traffic (0.0.0.0/0) to it. Instances also need public IPv4 addresses — auto-assigned via the subnet setting or attached as Elastic IPs — for the IGW to translate.

**4. How does a database server in a private subnet download patches without becoming reachable from the internet?**

*Answer:* Through a NAT gateway: it lives in a public subnet with an Elastic IP, and the private subnet's route table points internet-bound traffic at it. Outbound connections work; the internet can never initiate a connection inward.

**5. An instance sends a request out and the response never arrives. Its security group allows the outbound request. What's the difference in how the SG and a custom network ACL treat that response?**

*Answer:* The security group is stateful — the response is automatically allowed back in, whatever the inbound rules say. A network ACL is stateless — the response is fresh traffic and is dropped unless an inbound rule with a matching (lowest-number-first) allow exists.

**6. VPC A is peered with VPC B and with VPC C. Can B reach C through A? What else constrains peering?**

*Answer:* No — transitive peering is not supported; B and C need their own explicit peering connection. Peered VPCs also cannot have overlapping IP ranges, only one peering resource may exist per VPC pair, and both route tables need routes targeting the peering resource.

**7. You want to send 25% of users to a server running new code and 75% to the old one. Which Route 53 policy, configured how?**

*Answer:* Weighted round robin: two record sets on one DNS name with weights 1 and 3 (any values 0–255) — Route 53 returns the weight-3 record 75% of the time. This is the deck's A/B-testing use case.

**8. Distinguish a CloudFront edge location from a Regional edge cache, and explain why the second exists.**

*Answer:* Edge locations are the worldwide data centers serving popular content to viewers at lowest latency. When objects become less popular, edges evict them; Regional edge caches — larger caches placed between the origin and the edges — keep those objects nearby, cutting trips back to the origin and improving overall performance.