MODULE 3 · STUDY NOTES

Computing (Servers)

How Amazon EC2 delivers virtual servers — instance types and storage, the AMI-driven launch, networking and access, lifecycle and billing, and the purchasing options that fit each workload

Amazon Elastic Compute Cloud (Amazon EC2) is the servers of the AWS Cloud: virtual machines that you launch, configure, and tear down on demand instead of racking physical hardware. This module opens the hood on those instances. It starts with the virtualization model — how instances run on an AWS-managed hypervisor and receive virtual CPUs, memory, and either ephemeral instance-store or persistent Amazon EBS storage — then works through the choices you make every time you launch: which instance type, which Amazon Machine Image (AMI), which network and IP addressing, which block storage and security group, and which key pair. It covers the automation hooks (user data and instance metadata) and the identity mechanism (instance profiles carrying IAM roles) that let instances configure themselves and reach other AWS services without stored credentials. From there it turns to operating a fleet: the lifecycle states an instance moves through and what each means for billing, storage, and IP addresses; hibernation; resizing; keeping instances patched; the shared responsibility model that divides security duties between AWS and you; and finally the EC2 purchasing options that turn all of it into a cost decision.

LEARNING OBJECTIVES

- Describe Amazon Elastic Compute Cloud (Amazon EC2) virtualization
- Differentiate between the instance types and storage options available for EC2 instances
- Understand the networking components that must be specified when you launch an EC2 instance
- Explain Amazon EC2 user data and metadata
- Differentiate the lifecycle states for an EC2 instance
- Explain the shared responsibility model
- Create Amazon EC2 instances

3.1 Amazon EC2 and virtualization

Amazon EC2 instances run as virtual machines on host computers located in AWS Availability Zones. Each virtual machine runs an operating system — such as Amazon Linux or Microsoft Windows — on which you install applications, or even enterprise applications that span multiple virtual machines. The virtual machines run on top of a **hypervisor** layer that AWS maintains; the hypervisor is the operating-platform layer that gives each instance access to the actual hardware — processors, memory, and storage — it needs to run. Every instance receives a particular number of **virtual CPUs (vCPUs)** for processing and an amount of memory, or **RAM**.

INSTANCE STORE (EPHEMERAL)	AMAZON EBS (PERSISTENT)
physically attached to the host ▪ Temporary block-level storage attached to the host computer ▪ Data persists only for the lifetime of the instance ▪ **Survives a reboot** ▪ **Lost and unrecoverable** if the instance stops or terminates	*durable block storage* ▪ Persistent block-storage volumes for the boot disk and data ▪ Data persists even when the instance is **stopped** ▪ The default choice for most instances' boot and storage needs ▪ **EBS-optimized** instances cut I/O contention with other traffic

Instances also have **network connectivity** to other resources — other EC2 instances, Amazon Simple Storage Service (Amazon S3) object storage, and the internet at large — and you configure the degree of network access to balance accessibility against security requirements. Different instance types deliver different levels of network performance.

An instance type is a named bundle of virtual hardware. AWS offers a large selection so you can match the machine to the workload — some are general purpose, while others provide extra CPU (processing power), extra RAM (memory), or extra I/O network performance. The selection is a hierarchy: broad **categories**, then **families** (collections of types within a category), then individual **types** (unique selections of virtual hardware capability). A type's name encodes this — in `t3.nano`, `t3` is the family (in the General Purpose category) and `nano` is the specific type. The rule of thumb: once you know the workload's requirements, choose the **most cost-effective instance type** that supports them.

TABLE 3.1 Instance categories and what they emphasize

CATEGORY	EMPHASIS
General purpose	Balanced compute, memory, and networking (for example, the T3 family)
Compute optimized	Extra CPU / processing power
Memory optimized	Extra RAM for memory-intensive applications
Storage optimized	High-capacity, high-throughput local storage
Accelerated computing	Extra I/O and specialized hardware acceleration

> **WORKED EXAMPLE – TWO ENDS OF THE RANGE**
>
> The deck contrasts two real types. A `t3.nano` (General Purpose, T3 family) comes with **2 vCPU, 0.5 GB of memory, and up to 5 Gbps of network performance**, and its OS boots on Amazon EBS storage — a tiny, inexpensive machine. At the other extreme, an `x1e.32xlarge` is **memory optimized**, built for high-performance databases, in-memory databases, and other memory-intensive enterprise applications: **128 vCPU, 3,904 GB of memory, 14,000 Mbps of dedicated EBS bandwidth, 25-Gigabit network performance, and 3,840 GB of SSD storage**. Same service, radically different hardware — which is why matching the type to the workload matters for both performance and cost.

3.2 Launching an instance: AMIs and block storage

Every launch is a set of decisions layered on top of a template. You start with an **Amazon Machine Image (AMI)**, then specify the **instance type**, **network placement and addressing**, **block storage**, a **security group**, a **key pair**, and — optionally — **user data** and an **instance profile**. These are not an exhaustive list of options, but they are the items you

need to launch a secure, usable instance. Every instance is deployed into a network, either the legacy **EC2-Classic** or a **virtual private cloud (VPC)**.

An AMI is the template Amazon EC2 uses to launch an instance. You must specify a source AMI at launch, and you can launch **many identical instances** from one AMI when you need matching configurations — or use different AMIs for different configurations. You can also **create a new AMI from a running instance at any time**: launch three instances from one AMI, customize each differently, and capture each as its own new AMI to reuse as a template.

WHAT AN AMI CONTAINS – AND WHERE IT COMES FROM

- **Root-volume template** — an operating system, and optionally an application server and other software
- **Launch permissions** — which AWS accounts may use the AMI, including making it public
- **Block device mappings** — additional storage volumes to attach at launch
- Sources: **AWS-provided**, the **user community**, the **AWS Marketplace**, or **your own** AMIs

EXAM SCENARIO – THE FIRST STEP

The module's test-taking exercise asks: *what is the first step when you launch an EC2 instance?* The choices include selecting the instance type and size, providing configuration details, and setting access permissions — but a launch always begins with the **template**: **determine the AMI that will be used**. Everything else — type, network, storage, security group, key pair, user data — is layered on top of that choice.

When an instance uses Amazon EBS, you choose from several **EBS volume types**. The first split is technology: **SSD** volumes outperform **HDD** volumes, but HDD volumes cost less. Within each technology, AWS offers a higher-tier and a lower-tier option.

TABLE 3.2 Amazon EBS volume types

VOLUME TYPE	CLASS	BEST FOR
Provisioned IOPS SSD (io1)	SSD	Highest-performance EBS; I/O-intensive database and application workloads, and throughput-intensive databases/data warehouses (HBase, Cassandra)
General purpose SSD (gp2)	SSD	The **default** EBS type; broad transactional workloads, boot volumes, and low-latency interactive apps
Throughput optimized HDD (st1)	HDD	Frequently accessed, throughput-intensive workloads with large datasets and large I/O (MapReduce, Kafka, log processing, ETL)
Cold HDD (sc1)	HDD	**Lowest cost per GB** of all EBS types; less frequently accessed, large cold datasets

COMMON PITFALL

gp2 is the default, but it is not the ceiling. Choose `io1` (Provisioned IOPS SSD) when you need more IOPS than gp2 provides, when low latency is critical, or when you need better performance consistency. And to get the most from gp2 or st1, run them on **EBS-optimized** instances.

3.3 Networking, IP addressing, and security groups

You launch an EC2 instance into a network environment — typically a **VPC** created with Amazon Virtual Private Cloud (Amazon VPC), a logically isolated virtual network that closely resembles a traditional data-center network. Inside a VPC you define one or more **subnets**, logical network segments that each live in a **single Availability Zone**. An **internet gateway** — a horizontally scaled, redundant, highly available component — allows communication between VPC instances and the internet. An optional **virtual private gateway** supports VPN connections on the Amazon side; the customer side uses a **customer gateway** (a physical device or software application).

Three kinds of IP address attach to instances. A **private IP address** is always assigned at launch from the subnet's pool and lets instances communicate inside the VPC. A **public IP address** is optional, drawn dynamically from an AWS pool, and lets clients reach the instance from the internet — but it is *transient*: stop and start the instance and it gets a **new** public IP, though a plain **reboot keeps the same one**. An **Elastic IP address** is a public IP you provision that is **static**, and you can reassign it to another instance at any time. (A connection between two or more VPCs is a **peering connection**.)

A security group is a stateful virtual firewall applied at the instance level — not at the network's entry point — so different instances in the same subnet can carry different security groups. Every instance must have **at least one**; if you don't specify one at launch, it joins the VPC's **default** security group. Security groups control ICMP, TCP, and UDP traffic and restrict access by **port range**, **IP address / CIDR range**, or **source resource ID (another security group)**. A source of `0.0.0.0/0` (and `::0`) allows traffic from anywhere. You can attach **multiple** security groups to one instance, and one security group to many instances.

> **SECURITY GROUP RULES – GOOD TO KNOW**
>
> - **VPC** security groups have inbound *and* outbound rules; **EC2-Classic** security groups have inbound rules only
> - You cannot use an EC2-Classic security group on an instance running in a VPC
> - By default, a new VPC security group leaves **all outbound traffic open**
> - In a VPC you can **add or modify** security group associations while the instance is running

TABLE 3.3 Security group rule examples

GOAL	PORT	SOURCE
Allow HTTP from anywhere	80 (HTTP)	`0.0.0.0/0`
Allow SSH from one computer	22 (SSH)	`10.50.2.133/32`
Allow SSH from members of a group	22 (SSH)	`sg-4ad3712f`

The third pattern — allowing traffic only from instances that belong to a named security group — underlies the **bastion host** design: one hardened instance is reachable from outside the VPC, and it in turn is the only thing allowed to reach the private instances behind it.

3.4 Instance identity and automation

Applications on an instance often need to call AWS services — but **storing permanent access keys on an instance is not a security best practice**, because those credentials can be compromised. The answer is an **instance profile**, a container for an **IAM role** that you attach to the instance. The role carries one or more IAM policies that grant **temporary** access to account resources.

WHY AN INSTANCE PROFILE BEATS STORED KEYS

- **Automatic propagation** of access keys to the instance
- **Automatic rotation** of access keys multiple times a day
- Works across **multiple instances** — for example, an Auto Scaling group
- Example: grant instances permission to read the S3 buckets that hold application data, keeping data in S3 rather than on the instance

User data is a script that runs at launch — a shell script on Linux, or a batch or PowerShell script on Windows — that lets you deploy instances in an automated, repeatable way without ever logging in. It runs as a series of commands at the end of the first boot. The script can do a little or a lot: download and run a longer script from Amazon S3, install a configuration-management system such as Chef or Puppet, or call the AWS CLI (pre-installed on Amazon Linux). Scripts are executed by **cloud-init on Linux** and the **EC2Launch service on Windows**. By default user data runs **only on the first boot**, but you can configure it to run on every restart from a stopped state.

USER DATA ON LINUX	USER DATA ON WINDOWS
two forms ▪ A **shell script**, beginning with a `#!` (sha-bang) interpreter line ▪ **cloud-init directives**, beginning with `#cloud-config` ▪ cloud-config installs software the moment the instance first boots (may lengthen boot time)	*two forms* ▪ A **batch** script inside `<script>` tags — e.g., `winrm` calls to enable remote management ▪ A **PowerShell** script inside `<powershell>` tags — e.g., installing IIS as a web server ▪ Either way, configuration happens with no interactive login

Instance metadata is data about a running instance that you can use to configure or manage it. A metadata service is available to every instance at the link-local address `169.254.169.254`. Query `http://169.254.169.254/latest/meta-data/` and you get a listing of available items — `ami-id`, `hostname`, `instance-id`, `instance-type`, `security-groups`, `public-hostname`, `local-ipv4`, `placement/`, `iam/`, and many more. The user data itself is readable at `http://169.254.169.254/latest/user-data`.

WORKED EXAMPLE – METADATA INSIDE USER DATA

Properties such as the instance ID aren't known until launch, so a user-data script can query the metadata service to fill them in. The deck's example sets the machine's hostname to its own instance ID: it curls `http://169.254.169.254/latest/meta-data/instance-id`, writes that value into `/etc/hosts` and `/etc/sysconfig/network` with `sed`, and reboots so the change takes effect. You can also retrieve user data after the fact with `aws ec2 describe-instance-attribute --attribute userData` and `base64 --decode` the result.

3.5 Access, security, and launching instances

Amazon EC2 uses **public-key cryptography** for login: a public key encrypts data and the matching private key decrypts it — together they form a **key pair**. You typically name the key pair at launch (creating a new one or reusing an existing one), AWS configures the instance to accept it, and you then connect with the private key.

LINUX – SSH	WINDOWS – RDP
port 22 ▪ Connect with **SSH** using the key pair ▪ **Password-only authentication is disabled by default**	*port 3389* ▪ Connect with **Remote Desktop Protocol (RDP)** ▪ The key pair decrypts a randomly generated Administrator password

The default `ec2-user` (Linux) and Administrator (Windows) credentials are only a starting point — defining instance-level security and managing access as people join and leave is your job. For **Windows**, integrate with **AWS Directory Service**. For **Linux**, options include **Kerberos-based single sign-on (SSO)**, or generating per-user key pairs with `ssh-keygen` and adding each public key to the user's `~/.ssh/authorized_keys`. For a whole fleet, devise a strategy — baking users into AMIs is inflexible, so **configuration-management tools (Chef, Puppet, Ansible)**, which reduce granting and revoking access to a few commands, are the better answer.

A few options come into play at or after launch. The console can **capture a screenshot** of an instance to give visibility into its state. **Termination protection** guards against accidental deletion: it must be disabled before the instance can be terminated. And because every instance runs a **source/destination check** by default (it must be the source or destination of any traffic it handles), a **NAT instance** — which forwards traffic that is neither — requires that you **disable the source/destination check**.

> **LAUNCHING FROM THE AWS CLI**
>
> The `aws ec2 run-instances` command ties the whole module together in one line, with parameters for `--image-id` (the AMI), `--instance-type`, `--key-name`, `--security-group-ids`, `--subnet-id`, `--iam-instance-profile`, `--user-data file://UserData.txt`, and `--tag-specifications` to apply a Name tag. The `--subnet-id` is optional — omit it and the instance lands in the **default subnet of the default VPC**.

3.6 Managing instances: lifecycle, hibernation, and updates

An instance moves through a set of **lifecycle states**. It enters **Pending** when first launched or started; reaches **Running** once the OS has booted and it has passed a system status check and an instance status check; and can then be **Stopped** or **Terminated** (termination first stops, then terminates). One key limit: an instance whose root volume is an **instance store cannot be stopped — only terminated**; only **EBS-backed** instances can be stopped and started again. **Billing begins the moment an instance is Running.** A **reboot does not restart the billing cycle** and keeps the same public IP; a **stop-then-start begins a new billing hour immediately** and assigns a new public IP.

TABLE 3.4 Instance state – billing, storage, and IP effects

ASPECT	RUNNING	STOPPED	TERMINATED
Billing	Charged	Not charged	Not charged
Instance store	Retained	Lost	Lost
EBS volumes	Retained	Retained	Deleted if DeleteOnTermination is set
Public IP	Retained	Released	Released
Elastic IP	Retained	Retained	Disassociated

Hibernation takes a running instance and preserves its in-memory (RAM) data by persisting it to the EBS root volume, so on restart the instance returns to its previous state with processes resumed. It behaves like stop/start for addresses — the public IPv4 address is released and reassigned, while an Elastic IP is retained. You are **not charged for usage while hibernated**, but you **are** charged for the EBS storage, including the persisted memory contents.

> **HIBERNATION PREREQUISITES**
>
> - **Must be enabled at launch** — you cannot turn it on for an existing running or stopped instance
> - The root volume must be an **encrypted EBS** volume large enough to hold the RAM contents (the AMI must be encrypted)
> - Supported on specific families (C3–C5, M3–M5, R3–R5, T2) but excludes bare-metal (`.metal`) types
> - Available only for On-Demand, Spot, and Reserved Instances; instance-store-backed instances cannot be stopped or hibernated

Treat instances as **ephemeral** — things to build up, tear down, and rebuild on demand (for Auto Scaling, cost savings, downgrading, repairing an impaired instance, or upgrading the OS or image). **Resizing** an EBS-backed instance is simple: **stop it, modify its type, start it** (`aws ec2 modify-instance-attribute --instance-type m5.xlarge`), and the new type must share the **same architecture** (for example, 64-bit → 64-bit); an instance-store-backed instance must instead be **migrated** to a new instance. On the image side, **AMI deprecation does not affect running instances**: AWS keeps **Windows AMIs** for about three months (find them by name, not ID), while **Amazon Linux AMIs stay available for years**. Finally, keeping instances patched is your responsibility — Windows via **Windows Update**, Amazon Linux via **YUM** — with **AWS Systems Manager** and **AWS OpsWorks** available to manage updates at scale.

3.7 The shared responsibility model

Security on AWS is shared, and the shorthand is memorable: **AWS is responsible for security *of* the cloud; the customer is responsible for security *in* the cloud.** AWS manages and secures the underlying infrastructure — the facilities, the hardware, and the layers up to the hypervisor — and provides network and security monitoring, including DDoS protection and password brute-force detection on AWS accounts. Everything you put **on** that infrastructure is yours to secure, which means you must make several security decisions and configure the controls AWS provides.

AWS SECURES – *OF* THE CLOUD	CUSTOMER SECURES – *IN* THE CLOUD
the infrastructure ▪ Physical facilities, hardware, and the virtualization layer up to the hypervisor ▪ Network and security monitoring of the platform ▪ Baseline protections such as DDoS mitigation and account brute-force detection	*everything on the infrastructure* ▪ The EC2 instances and everything installed on them — including **guest-OS patching** ▪ The security groups that permit outside access to instances ▪ The VPC subnets instances live in, and external access to S3 buckets ▪ Correctly configuring the security tools AWS provides

> **COMMON PITFALL**
>
> Don't expect AWS to patch your instances. AWS has **no visibility into the OS layer**, so it cannot perform Windows Updates — or any guest-OS patching — for you. That is squarely the customer's responsibility under security *in* the cloud, everything up from the hypervisor.

3.8 Amazon EC2 purchasing options

How you **pay** for an instance is a separate decision from what the instance is, and AWS offers several purchasing options tuned to different usage patterns — from the maximum flexibility of On-Demand to the deep discounts of Reserved and Spot, plus dedicated and bare-metal hardware for licensing and compliance needs.

TABLE 3.5 Amazon EC2 purchasing options

OPTION	WHAT IT IS	BEST FOR
On-Demand	Pay per second or hour with no commitment; lowest upfront cost and the most flexibility	Short-term, spiky, or unpredictable workloads; apps in development or test
Reserved (RI)	A 1–3 year capacity reservation at a discounted hourly rate (up to 75% off); a billing construct, not a physical instance	Steady-state, predictable usage that needs reserved capacity
Scheduled RI	Capacity reserved to recur on a daily, weekly, or monthly schedule for a 1-year term	Jobs that run on a regular schedule for a finite time, not continuously
Spot	Spare capacity at 70–90% savings; AWS can reclaim it with a 2-minute notice	Fault-tolerant, urgent, or large-scale batch workloads
Dedicated Host	A physical server fully dedicated to you, with visibility and control over instance placement	Bringing per-socket / per-core / per-VM software licenses
Dedicated Instance	Instances that run on hardware dedicated to a single account	Regulatory requirements for physical isolation
Bare metal (i3.metal)	Direct access to the physical host's processor and memory, keeping cloud features (Auto Scaling, ELB, CloudWatch)	Non-virtualized workloads or hardware features like Intel VT-x

SPOT INSTANCE MECHANICS

- On interruption you choose the behavior: **hibernate** (saves memory state to disk, up to 122 GB), **stop**, or **terminate**
- AWS gives a **2-minute interruption notice**, delivered through the instance metadata
- Poll for the notice frequently (about every 5 seconds) and hook OS shutdown events to clean up
- Best practice: **encrypt the root EBS volume**, especially for instances that hibernate memory to disk

COMMON PITFALL

A Reserved Instance is **not a physical machine you are handed** — it is a capacity reservation that lowers your hourly rate. When a scenario calls for a *physical* dedicated server (for per-socket licensing or physical isolation), the answer is a **Dedicated Host** or **Dedicated Instance**, not a Reserved Instance.

Module summary

- Amazon EC2 instances are virtual machines on an AWS-managed hypervisor in Availability Zones; each gets vCPUs and RAM, with storage from ephemeral instance store (lost on stop/terminate, kept on reboot) or persistent Amazon EBS (kept on stop).
- Instance types are named hardware bundles organized as category → family → type; pick the most cost-effective type for the workload (tiny t3.nano versus memory-optimized x1e.32xlarge).
- Every launch starts from an AMI (root-volume image + launch permissions + block device mappings), then layers on the instance type, network and IP, block storage, a security group, a key pair, and optional user data.
- EBS volume types are SSD (io1 highest performance, gp2 the default general purpose) and HDD (st1 throughput-optimized, sc1 lowest cost per GB).
- Instances launch into a VPC with subnets (one AZ each), an internet gateway, and optional VPN gateways; private IPs are permanent, public IPs are transient (new on stop/start, same on reboot), and Elastic IPs are static.
- Security groups are stateful, instance-level firewalls that filter by port, IP/CIDR, or source security group; every instance has at least one, VPC groups have inbound and outbound rules, and associations can change while the instance runs.
- Attach an IAM role through an instance profile for temporary, auto-rotated credentials instead of stored keys; automate configuration with user data (cloud-init on Linux, EC2Launch on Windows) and read instance facts from the metadata service at 169.254.169.254.
- Lifecycle states drive billing and persistence: billing starts at Running, a reboot keeps billing and the public IP while stop/start resets both; hibernation must be enabled at launch on an encrypted EBS root volume.
- The shared responsibility model splits duties — AWS secures the cloud (infrastructure up to the hypervisor), and the customer secures in the cloud (instances, guest-OS patching, security groups, subnets, S3 access); AWS cannot patch your OS.
- Purchasing options match usage: On-Demand (flexible, no commitment), Reserved (up to 75% off for steady use), Scheduled RI (recurring jobs), Spot (70–90% off spare capacity, 2-minute notice), plus Dedicated Hosts/Instances and bare-metal i3.metal for licensing and compliance.

Review questions

1. **What happens to data on an instance store versus an EBS volume when an instance is stopped?**

   Answer: Instance-store data is lost — it is ephemeral and survives only a reboot — while EBS-volume data is retained because EBS is persistent block storage.

2. **List, in order, the main components you specify to launch a secure EC2 instance.**

   Answer: Start with the AMI, then the instance type, network placement and addressing (VPC/subnet and optional public IP), block storage (instance store or EBS), a security group, a key pair, and optionally user data and an instance profile.

3. **Name the four EBS volume types and which one is the default.**

   Answer: Provisioned IOPS SSD (io1), General purpose SSD (gp2), Throughput optimized HDD (st1), and Cold HDD (sc1); gp2 is the default.

4. **Why prefer an instance profile over access keys on an instance, and what does it contain?**

   Answer: Storing permanent keys risks compromise; an instance profile is a container for an IAM role that supplies temporary, automatically rotated credentials, which propagate to the instance and work across a fleet such as an Auto Scaling group.

5. **How does a reboot differ from a stop-and-start for billing and the public IP address?**

   Answer: A reboot does not restart the billing cycle and keeps the same public IP; a stop-and-start begins a new billing hour immediately and assigns a new public IP.

6. **What are the prerequisites and timing constraints for EC2 hibernation?**

   Answer: It must be enabled at launch (not on an existing instance), the root volume must be an encrypted EBS volume large enough for the RAM contents, and it is limited to specific families (excluding bare metal) and to On-Demand, Spot, and Reserved Instances.

7. **In the shared responsibility model, who applies operating-system patches to an EC2 instance, and why?**

   Answer: The customer — security in the cloud covers the guest OS and everything installed on the instance, and AWS has no visibility into the OS layer, so it cannot patch it.

8. **Match each situation to a purchasing option: an unpredictable dev/test workload; steady around-the-clock production; a fault-tolerant batch job that tolerates interruption; a need for per-socket software licensing.**

   Answer: On-Demand; Reserved Instance; Spot Instances; a Dedicated Host.