MODULE 5 · STUDY NOTES

Adding a Compute Layer Using Amazon EC2

Choosing the AMI, instance type, storage, and purchasing model that optimize the performance, security, and cost of an EC2-based compute layer

Amazon EC2 is the workhorse of AWS compute — resizable virtual servers you provision in minutes and pay for only while they run. This module takes the architect's view. It first places EC2 among the AWS compute categories (VMs, containers, VPS, PaaS, serverless) and dissects how an instance is virtualized on a host, then works through every decision that shapes an EC2-based compute layer: the **AMI** you launch from, the **instance type** that sizes CPU, memory, storage, and network, the **storage** you attach (instance store, EBS, EFS, or FSx), the **user data** and placement options that configure and position instances, and the **purchasing model** that decides what all of it costs. It closes by mapping those choices onto the **AWS Well-Architected Framework**, because the right compute layer is the one that is secure, performant, cost-optimized, and sustainable for a specific workload. The recurring theme: work backward from the business need, and expect the decisions to evolve as the workload does.

LEARNING OBJECTIVES

- Identify how to use Amazon Elastic Compute Cloud (Amazon EC2) in an architecture
- Explain the value of using Amazon Machine Images (AMIs) to accelerate the creation and repeatability of infrastructure
- Recommend EC2 instance types based on requirements
- Recommend storage solutions for Amazon EC2
- Recognize how to configure Amazon EC2 instances with user data
- Describe EC2 pricing options and make recommendations based on cost
- Launch an Amazon EC2 instance
- Use the AWS Well-Architected Framework principles when designing a compute layer with Amazon EC2

5.1 Adding compute with Amazon EC2

AWS offers several compute services because different workloads have different needs. The main runtime compute choices group into **five categories**: **virtual machines (VMs)**, **containers**, **virtual private servers (VPS)**, **platform as a service (PaaS)**, and **serverless**. The categories differ in how much of the infrastructure you manage versus how quickly you can deploy: VMs, containers, and VPS give **more infrastructure control and customization**, while PaaS and serverless let you **focus on the application** and deploy faster. This module focuses on the VM category and its service, **Amazon EC2**.

TABLE 5.1 AWS runtime compute choices

CATEGORY	AWS SERVICES	CONTROL VS. SPEED
Virtual machines (VMs)	Amazon EC2	Most infrastructure control and customization
Containers	Amazon ECS, Amazon EKS	Run Docker or Kubernetes apps at scale, highly available
Virtual private servers (VPS)	Amazon Lightsail	Simple management; low, predictable monthly price
Platform as a service (PaaS)	AWS Elastic Beanstalk	Upload code; the service handles deployment
Serverless	AWS Lambda, AWS Fargate	No servers to manage; fast deployment, built-in scaling and fault tolerance

Amazon EC2 provides virtual servers — *instances* — in the cloud. You provision servers in minutes, completely control their computing resources, scale capacity up or down (and resize existing servers) as requirements change, and **pay only for the capacity you use**. EC2 supports a wide range of operating systems including Microsoft Windows and many Linux variants, and it offers **Mac instances** that natively run macOS. It is the broadest and deepest compute platform in the cloud, with choices of processor, storage, networking, OS, and purchase model to match a workload.

An EC2 instance is a **VM running on a host computer** located in an Availability Zone. Each VM runs an operating system on which you install applications — even enterprise applications that span multiple VMs. The VMs sit on top of a **hypervisor** layer maintained by AWS that grants each instance access to the physical processors, memory, and storage it needs. Some instances use an **instance store** (ephemeral block storage physically attached to the host); many use **Amazon EBS** for the boot disk and other needs, which persists data even when the instance is stopped. **EBS-optimized** instances get faster volume access by minimizing I/O contention with other instance traffic. Instances also have configurable network connectivity to other instances, AWS services, and the internet, balanced against security requirements.

WHEN TO REACH FOR AMAZON EC2

- **Complete control of your computing resources** — configure everything from the OS to applications; choose x86 or ARM (AWS Graviton) processors, or accelerators such as machine learning chips.
- **Options for optimizing compute cost** — On-Demand, Reserved, Spot, and Savings Plans, plus Dedicated Hosts for dedicated capacity.
- **Ability to run any type of workload** — from simple websites and web/app/database/media servers to enterprise applications and generative AI.

Launching an instance is a sequence of configuration choices. You start from an **AMI** (the template), select an **instance type**, and — if you will connect over SSH or RDP — specify a **key pair** (a public/private credential pair). You choose **network placement** inside a **VPC** and whether to assign a public IP or DNS address; you assign a **security group** (a set of firewall rules defining which ports traffic may use); you set the **storage** the OS boots from (instance store or EBS) plus any additional volumes; if the application makes AWS API calls, you attach an **IAM role** (passed to the instance through an *instance profile*); and you can optionally supply **user data** to automate installation and configuration at launch.

COMMON PITFALL

EC2 is not the only compute answer. VMs, containers, and VPS trade convenience for control; PaaS and serverless trade control for deployment speed and less operational overhead. Start from the business case and work backward — choosing EC2 for a workload that Lambda or Elastic Beanstalk would serve more cheaply and simply is a design smell.

5.2 Choosing an AMI to launch an EC2 instance

You must specify a source **AMI** when you launch an instance. An AMI provides the information needed to launch it: a **template for the root volume** (the guest OS and any installed software), **launch permissions** (which AWS accounts can use the AMI, and whether it is public), and **block device mappings** (which additional storage volumes to attach). Amazon EC2 copies the template to the new instance's root volume and starts it. You can launch **many instances from one AMI** when you need identical configurations, or use **different AMIs** for different roles — one for web servers, another for application servers.

WHY AMIS MATTER – THE THREE RS

- **Repeatability** — an AMI packages the full configuration and content of an instance, so it launches multiple instances with efficiency and precision.
- **Reusability** — instances launched from the same AMI are exact replicas, making it easy to build clusters of similar instances or recreate environments.
- **Recoverability** — recreate a failed instance by launching from the same AMI, and back up a full instance configuration. Best practice: after changing an instance, save a **new AMI** so those changes are recoverable.

Choose an AMI on **five characteristics**: **Region** (each AMI exists in a specific Region; you can copy one across Regions), **operating system** (Windows or a Linux variant for AWS-provided AMIs), **root-device storage type** (Amazon EBS-backed or instance store-backed), **architecture** (32-bit or 64-bit, x86 or ARM), and **virtualization type** (paravirtual, PV, or Hardware Virtual Machine, HVM — **use HVM for best performance**). AMIs come from **four sources**: **Quick Start** (built by AWS — Windows or Linux variants such as Amazon Linux, Ubuntu, RHEL, SUSE), **My AMIs** (ones you create), **AWS Marketplace** (pre-configured third-party templates), and **Community AMIs** (shared by others, **not vetted by AWS — use at your own risk**, and avoid in production).

TABLE 5.2 Instance store-backed versus Amazon EBS-backed

CHARACTERISTIC	AMAZON EBS-BACKED	INSTANCE STORE-BACKED
Boot time	Boots faster	Slower — image parts fetched from Amazon S3
Max root device size	16 TiB	10 GiB
Stop the instance	Can be stopped	Cannot stop — only reboot or terminate
Change instance type	Yes — stop, then change	No — can't be stopped, so can't change
Charges	Instance usage + EBS volume + AMI as an EBS snapshot	Instance usage + AMI stored in Amazon S3 (usually cheaper)
Use case	Persistent storage	Temporary storage

The **instance lifecycle** turns on these differences. A launched (or restarted) instance enters **pending** while it is provisioned and boots, then **running**, where you can connect to it. From running, you can **reboot** (stays on the same

host, keeps its public DNS/IP and any instance-store data), or **terminate** (passes through *shutting-down* to *terminated*; the VM is eventually deleted and cannot be recovered). **EBS-backed** instances can also **stop** (through *stopping* to *stopped*) — a stopped instance can be started again, typically on a new host with a new public IPv4 address — and can **hibernate**, which saves the in-memory state, private IP, and Elastic IP so you resume where you left off. You are **not charged for the instance** while it is stopped or hibernated (though its EBS storage is still billed).

You **create a new AMI from an EC2 instance**. Start with a source AMI (an existing Quick Start AMI, or one built from a VM imported through the **VM Import/Export** service). Launch an EC2 instance from it (step 1) to get an *unmodified* instance, configure it into a *golden* instance with the OS and application settings you want (step 2), then capture it as a new AMI (step 3). For an **EBS-backed AMI**, creating the image auto-registers it; for an **instance store-backed AMI**, you bundle the root volume with the Amazon EC2 AMI tools, upload it to Amazon S3, and register it manually. Once registered, the AMI launches new instances in the same Region and becomes a new starter AMI — optionally **copied to other Regions** (step 4) to launch instances there.

EC2 Image Builder is another way to create AMIs. It automates the creation, management, and deployment of up-to-date, compliant golden VM images through a graphical pipeline interface, producing AMIs for AWS and VM images for on-premises use. It lets you build images with only the essential components (reducing exposure to security vulnerabilities), validate images with AWS-provided or your own tests before production, and enforce version control for revision management.

> **WORKED EXAMPLE – DUPLICATING A WEBSITE TO ANOTHER REGION**
>
> In the café challenge lab, the website is already running well on an existing EC2 instance and must also run in the Oregon Region. Because the instance is healthy, you **create an AMI from it**, **copy the AMI to the Oregon Region**, and **launch a new instance from that AMI in Oregon**. Remember what the AMI does *not* carry: when you launch the copy you still have to configure the **security group's rules**, choose the **instance type**, and attach the **IAM role** — those are separate from the AMI.

> **COMMON PITFALL**
>
> Unsaved changes are unrecoverable. If you add software or change configuration on an instance and do not capture a **new AMI**, a failure loses those changes — the old AMI only restores the old state. And never trust a **Community AMI** in a corporate or production environment; it is not vetted by AWS.

5.3 Selecting an EC2 instance type

Before launching, know the workload that will run on the instance — that drives the **instance type**, which defines a configuration of **vCPU, memory, storage, and network performance**. As the size of a type increases, so do its vCPU, memory, and storage; network performance scales more coarsely. The choice depends on your workload's **performance and cost** requirements. (All current-generation types support enhanced networking except **T2**.)

TABLE 5.3 How one family scales across sizes (m5d)

INSTANCE TYPE	VCPU	MEMORY	INSTANCE STORAGE	NETWORK
m5d.large	2	4 GiB	1 × 50 NVMe SSD	Up to 10 Gbps
m5d.xlarge	4	8 GiB	1 × 100 NVMe SSD	Up to 10 Gbps
m5d.8xlarge	32	128 GiB	2 × 600 NVMe SSD	10 Gbps

Instance type **names follow a convention**. Take `c7gn.xlarge`: `c` is the **family**, `7` is the **generation** (higher generations are generally more powerful and a better value), `gn` is an optional part indicating **processor family and additional capabilities** (here `g` = AWS **Graviton** processor, `n` = **network and EBS optimized**), and after the period `xlarge` is the **size**. Not every name includes the processor/capabilities part — `m5d.xlarge`, for example, has no processor-family letter.

TABLE 5.4 Instance type categories and suitable workloads

CATEGORY	OPTIMIZED FOR / WORKLOADS	EXAMPLE TYPES
General purpose	Balanced compute, memory, and networking — web/app servers, enterprise apps, gaming servers, dev/test	M7, M6, M5, M4, T4, T3, T2, Mac
Compute optimized	High-performance processors for compute-bound work — batch processing, distributed analytics, HPC, video encoding	C7, C6, C5, C4
Storage optimized	High sequential read/write to large local datasets; tens of thousands of low-latency IOPS — high-performance/NoSQL databases, real-time analytics, transactional, data warehouses	I4, I3, Im4, Is4, D2, D3, H1
Memory optimized	Fast performance for large in-memory datasets — in-memory caches, high-performance databases, big data analytics	R7, R6, R5, R4, X2, X1, Z1
Accelerated computing	Hardware accelerators/GPUs for floating-point, graphics, and pattern matching — machine learning and AI, HPC, graphics	P5, P4, P3, Inf2, Inf1, Trn1, DL1, G5, G4, F1, VT1
HPC optimized	Best price-performance for large-scale HPC — deep learning, compute-intensive HPC simulations	Hpc7, Hpc6

With **more than 270 instance types**, how do you choose? Aim for the type that meets your workload's performance needs while using resources efficiently to minimize cost. Analyze or anticipate the workload; the best type and size tend to reveal themselves during development. It is better to **slightly under-spec and then resize or scale** than to start with an oversized instance — an oversized instance is not cost-effective and can even obscure performance issues. AWS provides two tools: for a **new** instance, the **Instance Types page** in the EC2 console lets you search and filter by attribute (and pick the latest generation in a family for better price-to-performance); for a **running** instance, **AWS Compute Optimizer** recommends how to optimize the type.

AWS COMPUTE OPTIMIZER

- Recommends the optimal **instance type, instance size, and Auto Scaling group configuration** by analyzing configuration and utilization metrics.
- Uses machine learning; currently generates recommendations for the **M, C, R, T, and X** instance families.
- Classifies EC2 findings as **Under-provisioned, Over-provisioned, Optimized, or None**.
- **None** can occur if Compute Optimizer was activated for less than 12 hours, the instance has run less than 30 hours, or the type is unsupported.

ACTIVITY – MATCHING WORKLOADS TO INSTANCE FAMILIES

Given a set of workloads, pick the best-suited family: **transactional databases → I** (storage optimized), **small development environments → T** (general purpose), **gaming servers → M** (general purpose), **in-memory caches → R** (memory optimized), **image and video generation → Inf2** (accelerated computing), **machine learning → P** (accelerated computing), and **batch processing → C** (compute optimized). AWS recommends reviewing the Instance Type Details page frequently — new types are released that can improve efficiency and cost for existing workloads.

COMMON PITFALL

Do not default to the largest instance. Right-sizing means matching CPU, memory, storage, and network to the real workload — over-provisioning wastes money and can mask performance problems. When comparing within a family, the **newest generation** usually delivers the best price-to-performance, so favor it over an older, larger type.

5.4 Adding storage to an Amazon EC2 instance

EC2 instances have **four storage options**: **instance store**, **Amazon EBS**, **Amazon EFS**, and **Amazon FSx for Windows File Server**. Every instance always has a **root volume** and can optionally have one or more **data volumes**. Only an instance store or an **SSD-backed EBS** volume can serve as the root volume. For a **single-instance data volume**, use instance store or EBS (an EBS volume attaches to one instance at a time but can be detached and moved; an instance store volume is usable only by the instance it launched with). To **share a data volume across multiple instances simultaneously**, use Amazon EFS (Linux) or Amazon FSx for Windows File Server (Windows).

TABLE 5.5 EC2 storage resources by use case

STORAGE RESOURCE	ROOT	SINGLE-INSTANCE DATA	MULTI LINUX	MULTI WINDOWS
Amazon EBS (SSD-backed for root)	Yes	Yes	No	No
Instance store	Yes	Yes	No	No
Amazon EFS (Linux)	No	No	Yes	No
Amazon FSx for Windows File Server	No	No	No	Yes

Instance store provides temporary, non-persistent block-level storage on disks physically attached to the host that runs the instance. It works well for frequently changing temporary data — **buffers, caches, scratch data**. Volumes are exposed as block devices on HDDs or SSDs (SATA or NVMe; NVMe delivers higher I/O). The instance type determines available instance-store sizes and hardware, and most (not all) types support it. Data is **preserved on reboot but lost on terminate** — and because instance store-backed instances **cannot be stopped** (only rebooted or terminated), stopping is never a way to keep the data.

Amazon EBS provides network-attached, **persistent** block-level storage — like an external hard drive for an instance — that is highly available and reliable. Volumes attach to running instances **in the same Availability Zone**, **persist independently of the instance's life**, can be **encrypted**, and support point-in-time **snapshots to Amazon S3**. EBS volumes reside on HDDs or SSDs. Because they are mounted on the instance, they offer extremely low access latency, which makes EBS a good fit for running a database on EC2.

TABLE 5.6 Amazon EBS volume types

VOLUME TYPE	CATEGORY	BOOT?	BEST FOR
General Purpose SSD (gp2)	SSD / IOPS	Yes	Default type; balances price and performance for most workloads, dev/test, low-latency interactive apps
Provisioned IOPS SSD (io1)	SSD / IOPS	Yes	Highest-performance SSD; mission-critical, I/O-intensive, low-latency or high-throughput databases
Throughput Optimized HDD (st1)	HDD / MiB/s	No	Low-cost; frequently accessed, throughput-intensive — streaming, big data, data warehouses, log processing
Cold HDD (sc1)	HDD / MiB/s	No	Lowest cost per GB; large, infrequently accessed cold datasets

SSD-backed volumes suit **transactional** workloads where the dominant measure is **IOPS**; HDD-backed volumes suit large **streaming** workloads where **throughput (MiB/s)** matters more. Certain instance types can be **EBS-optimized**, giving a **dedicated network connection** to attached EBS volumes that minimizes contention and raises I/O performance — dedicated bandwidth ranges from **425 to 14,000 Mbps** depending on the type (for a gp2 volume, roughly a 10 percent improvement over baseline). Optimization is **enabled by default** on types categorized as EBS-optimized and **manually enabled** on other supporting types; instances built on the **AWS Nitro System** gain additional performance.

For data shared **among multiple instances at once**, neither an EBS volume (one instance at a time) nor Amazon S3 (an object store that overwrites whole files, not blocks) is ideal — you want the read/write consistency of a **network file system**. **Amazon EFS** provides a shared file system for **Linux** instances, and **Amazon FSx for Windows File Server** provides one for **Windows** instances. The available choice depends on the instance's operating system.

AMAZON EFS (LINUX)	AMAZON FSX FOR WINDOWS FILE SERVER
elastic shared file system for Linux	*native shared file system for Windows*
▪ Fully managed and elastic — scales automatically to petabytes as files are added and removed	▪ Fully managed native Windows file server built on Windows Server
▪ Uses Network File System (NFS) v4.x; mounts to the EC2 instance	▪ NTFS file system; SMB protocol 2.0–3.1.1; DFS Namespaces and Replication
▪ Compatible with all Linux-based AMIs; many NFS clients can access it concurrently	▪ Integrates with Microsoft Active Directory and supports Windows ACLs
▪ Standard class: a mount target per Availability Zone; One Zone: a single mount target, up to 47% cheaper	▪ SSD-backed for high throughput, high IOPS, and low latency

To reach an EFS file system inside a VPC you create **mount targets**. A mount target provides an **IP address for an NFSv4 endpoint**; instances mount the file system using its **DNS name**, which resolves to the mount target in the same Availability Zone. With a **Standard** storage class you can create a mount target in **each Availability Zone** (best durability and availability); with a **One Zone** storage class you create **one** mount target in the file system's AZ (cheaper, for less-frequently accessed data that doesn't require the highest resilience — accessing it from another AZ incurs data-access charges). Access the file system from a mount target in the **same AZ** as the instance for the best performance and cost.

MOUNTING EFS – THE LAB DETAILS

In the guided EFS lab you create a file system, log in to an Amazon Linux instance, and mount the file system with a command like `sudo mount -t nfs4 mount-target-DNS:/ ~/efs-mount-point`. Two facts to carry forward: instances that mount an EFS file system must be in the **same Region** (and within one VPC at a time), though they can span Availability Zones; and the mount target's **security group must allow inbound TCP on port 2049** for NFS.

COMMON PITFALL

Don't force the wrong storage onto a shared-access requirement. A lone EBS volume can't be attached to many instances at once, and Amazon S3 (object storage) rewrites entire files rather than blocks — poor for high-throughput file editing. Reach for **EFS on Linux** and **FSx for Windows File Server on Windows** when multiple instances need the same file system.

5.5 Other EC2 configuration considerations

When you launch an instance you can pass **user data** — an initialization script (shell commands or **cloud-init** directives) that the OS runs **once, at first launch**, with root or Administrator privileges, before the instance is accessible on the network. User data is a powerful way to automate setup: patch and update software from the AMI, fetch and install license keys, or install additional software. A typical Linux script begins with `#!/bin/bash`, then runs `yum update -y` to patch packages, `yum install httpd` to install the Apache web server, `service httpd start` to start it, and `chkconfig httpd on` so it starts on every boot.

The open-source **cloud-init** package bootstraps Linux instances (Amazon Linux, Ubuntu, and others include a version) and configures aspects of a new instance even without your directives — most notably the `.ssh/authorized_keys` file for the `ec2-user`, so you can log in with your private key. You can add your own cloud-init directives as user data. Script output is logged to `/var/log/cloud-init-output.log` on Linux (and a corresponding EC2Launch log on Windows). On Windows instances, user data is processed by **EC2Launch** (Windows 2016 and later) or the older **EC2Config**, which run PowerShell scripts.

To finish launching, user data often needs **instance metadata** — data about the instance itself, such as its public IP, private IP, hostname, instance ID, security groups, Region, and Availability Zone. Retrieve it from within the running instance at `http://169.254.169.254/latest/meta-data/` (a **link-local** address valid only from the instance); the user data itself is at `http://169.254.169.254/latest/user-data`. The service exposes much of the same information you would find in the console, and its simplicity — a plain HTTP request to a standard URL — is a key benefit.

RE-RUNNING USER DATA ON A RUNNING INSTANCE

- **Step 1 — Stop** the instance (this requires an EBS-backed instance).
- **Step 2 — Edit** the user data (in the console: Actions → Instance Settings → Edit User Data).
- **Step 3 — Delete** the `config_scripts_user` semaphore file under the instance's cloud-init `sem` directory (connect via SSH or SSM; this can't be done in the console).
- **Step 4 — Re-run** by restarting the instance, or by running the `part-001` user-data script directly without restarting.
- **Skip step 3 and the modified script won't run** — the delete is what re-arms it.

If you instead run additional configuration commands manually, it is a **best practice to update the user data script** anyway. Doing so keeps a **record of the instance's configuration** (a kind of version control) and lets you **copy the user data to new instances** that need the same setup. In the best-practice case, Web server A's manual changes are also

written back into its user data, so a new server copied from A is fully configured; if the script is left stale, copying it won't reproduce the current configuration.

TABLE 5.7 AMI deployment models

MODEL	CONFIGURATION	TRADE-OFFS	WHEN TO USE
Basic (base)	OS-only; configure via manual steps or user data	Fully configurable and upgradeable; shorter build, slower boot	As the base for building silver or golden AMIs
Silver (mutable)	Prerequisite software half-baked in; the rest via manual steps or user data (AMS-provided)	Balance of boot speed and build time; easier rollbacks; longer shelf life	One AMI reused for different purposes
Golden (immutable)	Fully baked — applications and all dependencies pre-installed	Shorter boot, longer build; shorter lifespan; every instance behaves identically	Fleets that must perform the same function

Finally, **placement groups** give you control over where a group of interdependent instances run within an Availability Zone (by default EC2 spreads instances across hardware to minimize correlated failure). Benefits are **higher network performance** between instances and **reduced correlated or simultaneous failure**. Limits: an instance can be in **only one placement group at a time**, and instances with a tenancy of *host* can't be launched into one. Three strategies: **Cluster** (packs instances close together for low-latency performance), **Partition** (spreads instances across logical partitions that don't share underlying hardware), and **Spread** (places a small group on distinct hardware to reduce correlated failures).

> **COMMON PITFALL**
>
> User data does not run on every reboot — only at first launch. Editing it on a live instance changes nothing by itself; you must stop, edit, **delete the config-scripts semaphore file**, then restart or re-run the part-001 script. And remember instance metadata lives at the link-local `169.254.169.254` address that is reachable only from inside the instance.

5.6 Amazon EC2 pricing options

The **AWS Free Tier** includes a 12-month EC2 offer for new customers: **750 hours per month** of a `t4g.small` instance (Region-dependent), plus 750 hours per month of a Linux/RHEL/SLES `t2.micro` or `t3.micro`, and 750 hours per month of a Windows `t2.micro` or `t3.micro`. When the 12 months end or usage exceeds the tier, standard pay-as-you-go rates apply. Beyond the Free Tier, EC2 pricing spans three groups: **purchase models** (big savings for different use cases), **capacity-reserved models** (guaranteeing capacity when you need it), and **dedicated models** (dedicated hardware for compliance and regulatory needs).

TABLE 5.8 Amazon EC2 purchase models

MODEL	HOW YOU PAY / COMMIT	RECOMMENDED FOR
On-Demand	Per second or per hour; no commitment; lowest upfront cost, most flexibility	Spiky, short-term, or unpredictable workloads; first-time testing
Reserved	1- or 3-year commitment; significant discount off On-Demand; fixed usage price	Predictable, steady-state workloads run for months or years
Savings Plans	Commit to a consistent spend ($/hour) for 1 or 3 years; Reserved-level discount, more flexibility	All EC2 workloads, especially those wanting flexibility with committed usage
Spot	Bid on spare capacity for substantial savings; 2-minute interruption notice	Fault-tolerant, flexible (non-time-critical), stateless workloads

Savings Plans come in two forms. **Compute Savings Plans** offer the most flexibility (up to **66 percent** savings) and apply automatically regardless of instance family, size, Availability Zone, Region, OS, or tenancy — you can shift between families, move a workload from one Region to another, or even move it to AWS Fargate or AWS Lambda and keep the Savings Plan price. **EC2 Instance Savings Plans** are less flexible but give the **largest discount** (up to **72 percent**), applying to a **specific instance family within a specific Region** while still letting you change instance size. Savings Plans cover EC2, AWS Lambda, and AWS Fargate usage.

Spot Instances let you use spare EC2 capacity at substantial savings, but your workload must tolerate interruption: EC2 can reclaim a Spot Instance with a **2-minute notification**. A Spot Instance runs while capacity is available and your maximum price exceeds the current Spot price. **Billing** is per-second with a 60-second minimum on **Amazon Linux or Ubuntu**; all other operating systems bill per hour, rounded up. A variation, **Spot blocks**, runs a workload continuously for a finite **1–6 hour** duration without interruption.

CAPACITY-RESERVED MODELS

- **On-Demand Capacity Reservations** — reserve compute capacity in a specific Availability Zone for any duration, guaranteeing access when you need it; good for regulatory high-availability requirements, capacity assurance, and disaster-recovery strategies.
- **EC2 Capacity Blocks for ML** — reserve **GPU instances** (for example, P5) for a future date to run machine learning workloads; pay only for the compute time you need, with no long-term commitment.
- Recommended uses for Capacity Blocks: training and fine-tuning ML models, running experiments and prototypes, and planning for future demand surges.

DEDICATED INSTANCES	DEDICATED HOSTS
single-tenant hardware, per-instance billing	*a whole physical server, per-host billing*
▪ Run on hardware dedicated to a single customer; automatic instance placement	▪ A full physical EC2 server dedicated to you; targeted placement and host affinity
▪ Physically isolated at the hardware level from other AWS accounts	▪ Visibility of sockets, cores, and host ID; add capacity via an allocation request
▪ May share hardware with non-dedicated instances from the same account	▪ Bring your own server-bound licenses — Windows Server, SQL Server, SUSE
▪ Billed per instance: hourly usage fee + a per-Region fee charged once per hour	▪ Purchase On-Demand, with Reservation Pricing, or via Savings Plans

Set an instance's **tenancy** attribute to *dedicated* or *host* to run it as a Dedicated Instance or on a Dedicated Host. Both isolate the hosts that run your instances from those of other accounts and provide the same performance, security, and physical features. The general **cost-optimization guideline** is to **combine purchase options**: run known **steady-state** workloads on **Reserved Instances or Savings Plans**, handle **stateful, spiky** demand with **On-Demand**, and scale **fault-tolerant, flexible, stateless** work with **Spot** — remaining consistent where the workload is consistent and paying only for bursts where it bursts.

> **SAMPLE EXAM QUESTION, WORKED**
>
> *A workload will run at least 1 year uninterrupted in the same Region. It stays steady except for seasonal spikes, when the instance size may increase — but the instance family stays the same. Which pricing option is lowest cost?* Key phrases: **at least 1 year in the same Region**, **steady**, **family stays the same**, **lowest cost**. **Dedicated Instances** (A) are among the most expensive and add hardware isolation the scenario doesn't need. **Compute Savings Plans** (B) fit, but their cross-family/Region flexibility isn't required and they cost more. **On-Demand** (D) is wrong for a year-long steady workload. The answer is **EC2 Instance Savings Plans** (C): they allow changing instance size within the same family in a Region and give the lowest price in exchange for committed usage.

5.7 Applying the AWS Well-Architected Framework principles to compute

The **AWS Well-Architected Framework** has **six pillars**, each with best practices and questions to consider when architecting. Four are most relevant to a compute layer: **security**, **performance efficiency**, **cost optimization**, and **sustainability**. The point of this section is that the EC2 decisions throughout the module — AMI choice, instance type and size, storage type, user data automation, and pricing model — are exactly the levers these best practices ask you to pull.

TABLE 5.9 Well-Architected best practices for compute

PILLAR	BEST PRACTICE	HOW THIS MODULE'S EC2 CHOICES APPLY
Security	Automate compute protection; control traffic at all layers	EC2 Image Builder reduces vulnerabilities; user data automates setup; silver/golden AMIs lock down config; a security group on every instance controls ports (defense in depth)
Performance efficiency	Scale the best compute options; configure and right-size	Choose AMIs pre-configured with required software and root-storage type; pick the right instance type/size and storage type for the workload
Cost optimization	Select the correct resource type, size, and number; select the best pricing model	Right-size instances and storage; combine On-Demand, Reserved, Savings Plans, Spot, and reserved/dedicated capacity models
Sustainability	Use minimum hardware; use instance types with least impact; use managed services	Instance store vs EBS by real need; adopt efficient, purpose-built newer instance types; use managed services such as AWS Batch and AWS Outposts

Infrastructure protection (security) divides into network and compute protection: automate protective mechanisms — vulnerability management, attack-surface reduction, resource management — to save time and reduce human error, and control traffic for both inbound and outbound flows with a defense-in-depth approach anchored by **security groups** (stateful firewalls). **Compute and hardware** (performance efficiency) is about selecting and right-sizing the compute option so you improve performance, avoid over- and under-provisioning, and lower cost. **Cost-effective resourcing** (cost optimization) pairs right-sizing with picking the pricing model that fits each component. **Hardware**

and services (sustainability) reduces environmental impact by minimizing hardware, continually adopting more energy-efficient instance types, and offloading undifferentiated work to managed services. None of these lists is exhaustive — the full set of best practices lives on the Well-Architected Framework website.

> **COMMON PITFALL**
>
> Well-Architected is not a checklist you run once at the end. The optimal compute choice varies with application design, usage patterns, and configuration — and right-sizing is **iterative**, triggered by changing usage, AWS price drops, or new instance types. Revisit instance type, storage, and pricing decisions as the workload evolves rather than treating the first design as final.

Module summary

- Amazon EC2 delivers resizable virtual servers in minutes with pay-for-what-you-use pricing; among AWS compute options (VMs, containers, VPS, PaaS, serverless) it gives the most infrastructure control, while PaaS and serverless trade control for deployment speed.
- An EC2 instance is a VM on a host in an Availability Zone, running above an AWS-maintained hypervisor; it can use ephemeral instance store and/or persistent, network-attached Amazon EBS, and EBS-optimized instances get a dedicated connection to their volumes.
- Launching an instance means choosing an AMI and instance type and setting network/VPC, security group, storage, IAM role (via an instance profile), key pair, and optional user data.
- An AMI supplies the root-volume template, launch permissions, and block device mappings; its value is repeatability, reusability, and recoverability. Use HVM for best performance and source AMIs from Quick Start, My AMIs, AWS Marketplace, or Community (unvetted).
- EBS-backed instances boot faster, reach a 16 TiB root, and can stop and change type; instance store-backed instances cap at a 10 GiB root, can't stop or change type, and lose data on stop or terminate (they persist only across a reboot).
- Instance types set vCPU/memory/storage/network and fall into six categories (general purpose, compute-, storage-, memory-optimized, accelerated computing, HPC); newer generations offer better price-to-performance. Use the console Instance Types page for new instances and AWS Compute Optimizer for running ones.
- Root volumes use instance store or SSD-backed EBS; single-instance data uses instance store or EBS; shared file systems use Amazon EFS (Linux/NFS) or Amazon FSx for Windows File Server (Windows/SMB). EBS SSD types gp2/io1 can boot; HDD types st1/sc1 cannot.
- User data runs an initialization script once at first launch; instance metadata is read in-instance at the link-local 169.254.169.254 URL. AMI deployment models are Basic, Silver, and Golden, and placement groups (Cluster, Partition, Spread) control where interdependent instances run.
- EC2 pricing spans On-Demand (spiky), Reserved and Savings Plans (steady-state commitments), and Spot (fault-tolerant/stateless, 2-minute interruption); Dedicated Instances/Hosts give single-tenant hardware and Capacity Reservations (including Capacity Blocks for ML) assure capacity. Optimize by combining models.
- The Well-Architected Framework frames compute decisions: automate protection and control traffic (security), scale and right-size compute (performance efficiency), pick the right resource and pricing model (cost optimization), and minimize least-impact hardware plus managed services (sustainability).

Review questions

1. A workload must share one file system across a fleet of Linux instances in several Availability Zones. Which storage service, and what must you configure to reach it?

Answer: Amazon EFS (a Linux/NFS shared file system). Create a mount target in each Availability Zone (Standard class); instances mount via the DNS name, and the mount target's security group must allow inbound TCP 2049 (NFS).

2. Explain why an instance store-backed instance cannot be stopped, and what that means for your data.

Answer: Instance store is ephemeral storage physically attached to the host, so there is nowhere to preserve state while the instance is stopped — it can only be rebooted or terminated, and its data survives a reboot but is lost on termination. You also can't change its instance type. Use EBS-backed instances when you need to stop/start, change type, or persist data.

3. You need to launch identical web servers repeatedly and recover quickly if one fails. How do AMIs help, and what do they NOT capture?

Answer: An AMI packages the full instance configuration, so every launch is an identical replica (repeatability and reusability) and a failed instance is replaced by launching from the same AMI (recoverability). It does not capture the security group, IAM role, or instance type — set those at launch. Best practice: after changing an instance, save a new AMI.

4. Decode c7gn.xlarge, and state the general rule about generation numbers.

Answer: c = family, 7 = generation, g = AWS Graviton processor, n = network and EBS optimized, xlarge = size. Higher-generation instances in a family are generally more powerful and offer better price-to-performance.

5. Match each workload to an instance-family category: transactional database, in-memory cache, machine learning training, batch processing.

Answer: Transactional database → storage optimized (I family); in-memory cache → memory optimized (R family); machine learning → accelerated computing (P family); batch processing → compute optimized (C family).

6. A steady workload runs a year in one Region; seasonal spikes only increase instance size within the same family. Which pricing option is cheapest, and why not the alternatives?

Answer: EC2 Instance Savings Plans — they lock to one instance family in a Region for the largest discount while still allowing size changes. Compute Savings Plans add unneeded cross-family/Region flexibility at a smaller discount; On-Demand is wrong for year-long steady use; Dedicated Instances are among the most expensive and their hardware isolation isn't required.

7. You edited an EC2 instance's user data but the new script didn't run on restart. What step was missed?

Answer: User data runs only at first launch. To re-run it you must stop the instance, edit the user data, then delete the config_scripts_user semaphore file (via SSH or SSM) before restarting or running the part-001 script. Skipping the delete step means the modified script won't run.