MODULE 2 · STUDY NOTES

Introducing Cloud Architecting

What cloud architecture is, the role of the architect, the six-pillar AWS Well-Architected Framework, ten best practices for building on AWS, and the global infrastructure where resources live

Before there was cloud architecting, there was Amazon around the year 2000 — trying to build an ecommerce platform for third-party sellers and drowning in a code base that had grown without a plan. The fix, refined over years, was to build well-documented, reusable internal services for highly available, scalable, and reliable systems — and in 2006 Amazon began selling them as AWS. This module distills the discipline that grew out of that experience. Cloud architecture is the practice of applying cloud characteristics to a solution that meets an organization's technical needs and business use cases, and the cloud architect — like the architect of a building — turns a decision maker's requirements into a blueprint that a delivery team implements. To judge whether a design is sound, AWS offers the Well-Architected Framework: six pillars of foundational questions and best practices drawn from thousands of real customer architectures. On top of the framework sit ten concrete best practices for building on AWS, from scalability and automation to caching and security. And because every architecture ultimately runs somewhere, the module closes with the AWS Global Infrastructure — Regions, Availability Zones, Local Zones, data centers, and edge locations — and how to decide where your resources should live.

LEARNING OBJECTIVES

- Define cloud architecture
- Describe how to design and evaluate architectures using the AWS Well-Architected Framework
- Explain best practices for building solutions on Amazon Web Services (AWS)
- Describe how to make informed decisions about where to place AWS resources

2.1 Cloud architecting

Cloud architecting exists because building without it is painful. Around 2000, Amazon set out to build an ecommerce service that let third-party sellers run their own shopping sites on top of the Amazon ecommerce engine — and struggled to make it highly available and scalable. The early tools and architectures had been built without proper planning: services were hard to separate, development was slow, and projects were complicated. Organizing the environment behind a set of **well-documented APIs** helped, but as the company hired more engineers, new problems appeared — it could take three months just to build the database, compute, and storage components, and each team built its own resources with no thought for scalability or reusability. The lasting solution was to build **internal services** that produced highly available, scalable, and reliable architectures on top of Amazon's infrastructure. In 2006 Amazon began selling those services as **Amazon Web Services (AWS)**, starting with Amazon SQS, then Amazon S3 and Amazon EC2.

Cloud architecture is the practice of applying cloud characteristics to a solution that uses cloud services and features to meet an organization's technical needs and business use cases. Creating a technology solution is a lot like constructing a building: if the foundation is not solid, structural problems undermine everything above it. In construction, the customer outlines the requirements, the architect turns them into a design and blueprints, and the

building crew makes the blueprint physical. Cloud architecture follows the same roles — the customer or **decision maker** outlines business goals and requirements, the **cloud architect** designs a solution that acts as the blueprint, and the **delivery team** implements it. Well-architected systems increase the likelihood that the technology deliverables will actually help meet the business goals.

TABLE 2.1 The building analogy for a cloud project

STAGE / ROLE	CONSTRUCTING A BUILDING	CLOUD ARCHITECTURE
Requirements	Customer outlines the building needs	Decision maker sets business goals and requirements
Design	Architect creates the blueprint	Cloud architect designs the solution (the blueprint)
Delivery	Building crew constructs the structure	Delivery team implements the solution

WHAT A CLOUD ARCHITECT DOES

- **Plan** — set the technical cloud strategy with business leads, and analyze solutions against business needs and requirements.
- **Research** — investigate cloud service specifications and workload requirements, review existing workload architectures, and design prototype solutions.
- **Build** — design the transformation roadmap (milestones, work streams, and owners) and manage adoption and migration.
- Engage decision makers to identify business goals, ensure the technology deliverables align with those goals, and work with delivery teams so the features are appropriate.
- Develop the technical cloud strategy, assist with migration, review workload requirements, and give guidance on high-risk issues — all while working to implement the **AWS Well-Architected Framework**.

Whatever the task, a cloud architect **starts with the business case and works backward** to choose the best components for that use case — not the other way around. Design decisions are also expected to **evolve over time** to optimize cost and performance as needs change or become better understood. Along the way, AWS services let you create **highly available, scalable, and reliable** architectures, but which services you use and how they fit together is always driven by the business need.

COMMON PITFALL

Don't start from the technology. The recurring discipline of this module is to work backward from the **business need** to the architecture — choosing services to fit the use case — rather than picking a favorite service and forcing the problem to fit it.

2.2 AWS Well-Architected Framework

The AWS Well-Architected Framework is a guide that provides a consistent approach to evaluate cloud architectures and guidance to help implement designs. It documents a set of foundational questions and best practices you can use to judge whether a specific architecture aligns with cloud best practices — knowledge AWS distilled after reviewing **thousands of customer architectures**. The framework is organized into **six pillars**, and each pillar has its own whitepaper detailing its design principles and best practices. Most modules in this course revisit selected pillars as they relate to the material, so treat this section as the map you will return to again and again.

TABLE 2.2 The six pillars and their design principles

PILLAR	WHAT IT ADDRESSES	KEY DESIGN PRINCIPLES
Operational Excellence	Run and monitor systems to deliver business value; continually improve supporting processes and procedures	Deliver business value; continually improve; view the entire workload as code
Security	Protect information, systems, and assets while delivering value, through risk assessment and mitigation	Strong identity foundation; maintain traceability; security at all layers; risk assessment and mitigation
Reliability	Recover from infrastructure or service disruptions and dynamically acquire resources to meet demand	Recover quickly; dynamically meet compute demand; mitigate disruptions
Performance Efficiency	Use computing resources efficiently and maintain that efficiency as demand changes	Choose and maintain efficient resources; democratize advanced technologies; employ mechanical sympathy
Cost Optimization	Deliver business value at the lowest price point; an ongoing, iterative requirement	Measure efficiency; eliminate unneeded expense; adopt the right consumption model; use managed services
Sustainability	Build architectures that maximize efficiency and reduce waste; minimize downstream environmental impact	Establish sustainability goals; maximize utilization; choose efficient hardware and software; reduce downstream impact

A few pillar ideas deserve extra emphasis. **Operational Excellence** encourages you to view your *entire* workload — applications, infrastructure, policies, governance, and operations — **as code**, so you can apply the same engineering discipline to every element of the stack and set up automated responses. **Performance Efficiency** introduces **mechanical sympathy**: using a tool or system with an understanding of how it operates best (for example, considering data access patterns when you select a database or storage approach), and **democratizing advanced technologies** by letting a vendor absorb the complexity of hard-to-implement capabilities. **Cost Optimization** is an ongoing, iterative requirement — adopt a consumption model where you pay only for what you use, and prefer managed services that operate at cloud scale for a lower cost per transaction. **Sustainability** focuses on energy reduction and efficiency: efficient languages and algorithms, right-sized compute, and minimizing the downstream impact of your workloads.

The AWS Well-Architected Tool (WA Tool) puts the framework to work. It is a self-service tool, available in the **AWS Management Console**, that gives on-demand access to current AWS best practices. You define your workload and answer a series of questions in the areas of **operational excellence, security, reliability, performance efficiency, and cost optimization**; the tool then delivers an **action plan** with step-by-step guidance on how to improve the workload. It provides a consistent process to review and measure your architectures, and it is intended for the people involved in technical product development — CTOs, architects, developers, and operations team members.

WHAT THE WA TOOL DOES FOR YOU

- Reviews the state of your workloads and **compares them to the latest** AWS architectural best practices.
- Gives on-demand access to the knowledge and best practices AWS architects use.
- Delivers a **step-by-step action plan** for building better workloads for the cloud.
- Provides a **consistent process** for you to review and measure your cloud architectures.
- Gathers data and recommendations to minimize system failures and operational costs, dive deep into business and infrastructure processes, provide best-practice guidance, and deliver on the cloud value proposition — feeding architectural decisions into corporate governance.

COMMON PITFALL

The framework has **six** pillars, not five — Sustainability is the one students most often forget. (The WA Tool's question areas in this module cover the first five; Sustainability is still a full pillar of the framework.)

2.3 Best practices for building solutions on AWS

Every design is a set of trade-offs. As you design a solution, think carefully about trade-offs so you can select an optimal approach — for example, trading **consistency, durability, and space for time and latency** to deliver higher performance, or prioritizing **speed to market over cost** for a new feature. Because trade-offs can increase the cost and complexity of an architecture, base your decisions on **empirical data** rather than intuition: perform **load testing** to confirm a measurable performance benefit, or **benchmarking** to reach the most cost-optimal workload over time. This section walks through ten best practices for building on AWS — and, just as importantly, the **anti-patterns** (bad solution designs) each one helps you avoid.

TABLE 2.3 Ten best practices for building on AWS

BEST PRACTICE	CORE IDEA	ON AWS
Implement scalability	Handle changes in demand automatically at every layer; be elastic	CloudWatch alarms + EC2 Auto Scaling
Automate your environment	Auto-provision, terminate, and configure resources; respond to failures	CloudWatch + EC2 Auto Scaling; IaC
Treat resources as disposable	Think of infrastructure as software; replace rather than repair	Identical configs; stop idle resources
Use loosely coupled components	Put a managed intermediary between layers	Load balancers (ELB) and message queues
Design services, not servers	Don't limit the design to EC2 servers	Lambda, SQS, DynamoDB, ELB, SES, Cognito, S3
Choose the right database	Match the technology to the workload, not the reverse	Purpose-built AWS database services
Avoid single points of failure	Assume everything fails, then design backward	Standby replica; managed services
Optimize for cost	Trade fixed expense for variable; provision only what you need	Right-size; stop idle; managed services
Use caching	Store data in an intermediary to cut redundant retrieval	Amazon CloudFront in front of Amazon S3
Secure your entire infrastructure	Build security into every layer	Least privilege, MFA, encryption, security groups

KEY TERMS

Infrastructure as code (IaC)
Provisioning computing infrastructure with code instead of manual processes, so you can deploy duplicate environments from one template, reduce configuration errors, propagate changes consistently to all stacks, and roll back to a known-good version

Loose coupling
A design that puts a managed intermediary — a load balancer or a message queue — between the layers of a system so the intermediary handles failures and scaling automatically

Disposable resources
Treating infrastructure as software rather than hardware: deploy new resources from identical configurations, stop what you are not using, and replace old resources instead of repairing them

Anti-pattern
A bad solution design to avoid — such as configuration drift across servers, hardcoded IP addresses, or a single database that becomes a single point of failure

Scalability and automation work together. When your workloads run on AWS, you can scale infrastructure quickly and proactively — and you should implement scalability at **every layer**. A monitoring service like **Amazon CloudWatch** can detect when load crosses a threshold (say, CPU utilization staying above **60% for more than five minutes**) and, through **Amazon EC2 Auto Scaling**, launch a new instance *before* capacity is reached, giving users a seamless

experience; design the system to be **elastic** so that when demand drops you are not paying for instances you no longer need. Contrast that with reactive, manual scaling, where users are locked out while an administrator launches instances that take minutes to become available. **Automation** extends the same idea to provisioning, termination, and configuration: built-in tools can detect an unhealthy resource, launch a replacement, notify an administrator, and log the change automatically. A key enabler is **infrastructure as code (IaC)** — provisioning through code rather than manual steps to deploy duplicate environments, reduce human error, and push changes consistently across stacks.

TIGHTLY COUPLED (ANTI-PATTERN)	LOOSELY COUPLED
chains of servers, each with a fixed purpose ▪ Web servers connect directly to specific application servers ▪ If one component or layer goes down, the disruption to the system can be fatal ▪ Scaling forces you to reconnect every server on each adjoining layer	*a managed intermediary between layers* ▪ A load balancer (ELB) routes requests and redirects traffic to healthy servers on failure ▪ A message queue buffers communication between applications ▪ The intermediary handles both failures and the scaling of components automatically

Design services, not servers, and match the database to the workload. Amazon EC2 offers tremendous flexibility, but it should not be the first or only tool for every need — when appropriate, reach for **containers or a serverless solution**. Message queues can handle communication between applications, static web assets can live **off-server** on Amazon S3, and user authentication and state can be handled by managed services; examples of lower-profile, more performant replacements for server-based designs include **AWS Lambda, Amazon SQS, Amazon DynamoDB, ELB, Amazon SES, and Amazon Cognito**. The same fit-the-tool-to-the-job logic drives database choice: **match the technology to the workload, not the other way around.** AWS offers a large collection of purpose-built databases, so choose based on read and write needs, total storage, object size and access patterns, durability and latency requirements, concurrent users, the nature of your queries, and the required strength of integrity controls.

FIXED EXPENSE (TRADITIONAL)	VARIABLE EXPENSE (AWS)
pay whether or not you use it ▪ Funds spent to acquire, upgrade, and maintain physical assets — property, buildings, equipment ▪ You pay for data-center servers whether they are active or not ▪ Replicating a 24/7 on-premises setup in the cloud is very expensive	*pay only for what you use* ▪ Pay only for the individual services you need, for as long as you use them ▪ Each service can be optimized for cost with tiers, models, and configurations ▪ Provision only what you need and stop services when they are not in use

Assume everything fails, then design backward. Eliminating single points of failure does not require duplicating every component — depending on your downtime SLAs you can use automated solutions that launch components only when needed, or a managed service that lets AWS replace malfunctioning hardware for you. A common pattern is a **secondary (standby) database** with replicated data that takes over if the primary goes offline. And you should **secure your entire infrastructure** by building security into every layer: use managed services, log resource access, isolate parts of the infrastructure, encrypt data **in transit and at rest**, enforce granular access control with the **principle of least privilege**, require **multi-factor authentication (MFA)**, and automate deployments to keep security consistent. In Amazon EC2, for example, **security groups** determine which ports can send and receive traffic and where that traffic may come from or go — reducing the chance that a threat on one instance spreads across your environment.

WORKED EXAMPLE – CACHING WITH CLOUDFRONT AND S3

Without caching, every user request for a file goes all the way to the Amazon S3 bucket, so each request takes the same amount of time and costs the same. Add **Amazon CloudFront** in front of S3 and the first request checks CloudFront, misses, fetches the file from S3, and stores a copy at an **edge location** close to the user before returning it. The second, third, and nth requests are served from that nearby edge location instead of S3 — at **lower latency and lower cost** — and after the first request you no longer pay to transfer the file out of S3. That is the payoff of minimizing redundant data retrieval.

COMMON PITFALL

Amazon EC2 is not the default answer. Designing only for servers is an anti-pattern — simple applications pinned to persistent servers, applications talking directly to one another, static assets stored on instances, and backend servers handling authentication and state. **Design services, not servers**: use containers, serverless, queues, and managed data and identity services when they fit.

2.4 AWS Global Infrastructure

The AWS Global Cloud Infrastructure is where your architecture actually runs. It is a secure, extensive, and reliable platform offering more than **200 fully featured services** from data centers around the world, spanning — at the time of the course — **102 Availability Zones across 32 geographic Regions**. This footprint lets you launch highly available, scalable, and flexible architectures near your customers for lower latency and increased performance. As you design, you decide where to deploy, and the decision rests on your **business requirements and workload needs**. This section covers the building blocks: **Regions, Availability Zones, AWS Local Zones, data centers, and AWS points of presence (PoPs)**.

TABLE 2.4 AWS global infrastructure building blocks

ELEMENT	WHAT IT IS
Region	A physical geographic area of two or more Availability Zones; isolated from other Regions; connected over the AWS backbone network
Availability Zone	One or more data centers, designed as an independent failure zone, linked to other AZs by high-speed private links
Local Zone	An extension of a Region that places select services near large population, industry, and IT centers for single-digit-millisecond latency
Data center	Where data resides and processing occurs — typically tens of thousands of servers; you never select one directly
Edge location	A PoP close to customers that delivers content with the lowest possible latency (CloudFront, Route 53, Global Accelerator)
Regional edge cache	A PoP between the origin server and the edge locations, with a larger cache for less-frequently accessed content

Selecting a Region. A Region is a physical geographic area with two or more Availability Zones. Regions connect to multiple internet service providers (ISPs) and to a **private global network backbone**, which delivers lower-cost, more consistent cross-Region latency than the public internet. Two operational details matter: Regions introduced **before March 20, 2019 are enabled by default**, while those introduced after (such as Asia Pacific Hong Kong and Middle East Bahrain) are **disabled by default** and must be enabled from the console; and some Regions have **restricted access** —

the isolated **AWS GovCloud (US)** Regions serve US government workloads with specific regulatory and compliance needs. Most importantly, Regions are **isolated from one another** for fault tolerance and stability: resources in one Region are **not automatically replicated** elsewhere, so replicating data across Regions is **your responsibility**. Choose a Region based on **compliance and network latency** requirements, and remember that not every service is available in every Region.

Selecting Availability Zones. Each Region consists of two or more isolated locations called Availability Zones, and each AZ comprises **one or more data centers** — some with as many as six — though **no data center belongs to two AZs**. Every AZ is engineered as an **independent failure zone**: physically separated within a typical metropolitan Region, sited in lower-risk floodplains, powered by discrete uninterruptible power supplies and on-site backup generators fed by **different grids from independent utilities**, and redundantly connected to multiple **tier-1 transit providers**. AZs within a Region are interconnected by **high-speed private links**. For certain services, such as Amazon EC2, the AZ is the most granular placement you can specify — you select which AZs your systems reside in, and AWS recommends **replicating across AZs for resiliency**. Systems can span multiple AZs, and you should design them to survive the temporary or prolonged failure of one, keeping applications resilient through natural disasters or system failures.

Using Local Zones. A Local Zone is a type of AWS infrastructure deployment that places AWS compute, storage, database, and other **select services closer to large population, industry, and IT centers where no Region exists today**. Each Local Zone is an **extension of a Region**, connected back to it over a high-bandwidth, secure link and reachable through the same APIs and toolsets — so you can run latency-sensitive parts of an application (using services such as Amazon EC2, Amazon VPC, Amazon EBS, Amazon FSx, and ELB) close to end users while the rest runs in the Region. Local Zones are **managed and supported by AWS** and deliver **single-digit-millisecond latency** for use cases such as media and entertainment content creation, real-time gaming, reservoir simulation, electronic design automation, and machine learning — with the usual cloud elasticity, scalability, and pay-only-for-what-you-use pricing.

The role of data centers. Data centers are the foundation of the infrastructure — where data actually resides and processing occurs — and a data center typically houses **tens of thousands of servers**. You **do not specify a data center** for a deployment; you choose a Region and, for some services, an AZ. **All data centers are online and serving customers**, and in the rare event of a failure, automated processes move customer traffic away from the affected area. Core applications are deployed in an N+1 configuration, so there is enough spare capacity to load-balance traffic to the remaining sites. AWS uses **custom network equipment sourced from multiple original device manufacturers (ODMs)** — companies that design and build products to another company's specifications for rebranding — along with a customized network protocol stack.

AWS points of presence (PoPs) sit at the edge of the network to reduce latency, and there are two kinds. An **edge location** is a set of AWS data centers and servers located close to customers, designed to deliver content with the **lowest possible latency**; edge locations support services such as Amazon Route 53, AWS Global Accelerator, and Amazon CloudFront. A **regional edge cache** sits **between the origin server and the edge locations** and has a **larger cache**, absorbing content that is not accessed frequently enough to stay in an edge location and providing an alternative to fetching it from the origin. Amazon CloudFront uses a global network of more than **410 PoPs — about 400 edge locations and 13 regional mid-tier caches** — and regional edge caches are used by default, transparently to the end user.

DECIDING WHERE YOUR RESOURCES LIVE

- **Compliance first** — some data must stay in a specific country or Region; AWS publishes where each Region resides, and you are responsible for choosing accordingly.
- **Then latency** — deploy near your customers, and for the most latency-sensitive workloads reach further out with **Local Zones**, **edge locations**, and **regional edge caches**.
- **Design for AZ failure** — spread systems across **multiple Availability Zones** so they stay resilient through natural disasters or system failures.
- **Replicate across Regions only when you need to** — it is your responsibility, and Regions never replicate automatically.
- You choose a **Region** and (for some services) an **AZ** — never a specific **data center**.

COMMON PITFALL

Two facts trip students up: resources are **not** replicated across Regions automatically (that is your job), and you **cannot** choose a data center — only a Region and, for some services, an Availability Zone. Also remember that an AZ can contain **several** data centers, and Regions added after March 20, 2019 are **disabled by default**.

Module summary

- Cloud architecture applies cloud characteristics to a solution that meets an organization's technical needs and business use cases; like a building's architect, the cloud architect turns a decision maker's requirements into a blueprint the delivery team implements — always working backward from the business need.
- AWS grew out of Amazon's own struggle (circa 2000) to build highly available, scalable systems; the fix was reusable internal services, sold from 2006 as AWS (Amazon SQS, then S3 and EC2).
- The AWS Well-Architected Framework gives a consistent way to evaluate and improve architectures through foundational questions and best practices, organized into six pillars: Operational Excellence, Security, Reliability, Performance Efficiency, Cost Optimization, and Sustainability.
- The AWS WA Tool (in the Management Console) reviews a workload against current best practices across operational excellence, security, reliability, performance efficiency, and cost optimization, and returns a step-by-step action plan — it reviews and recommends, it does not remediate for you.
- Base design trade-offs on empirical data (load testing, benchmarking), not assumptions, and learn the anti-pattern each best practice avoids.
- The ten best practices: implement scalability, automate the environment (with IaC), treat resources as disposable, loosely couple components, design services not servers, choose the right database, avoid single points of failure, optimize for cost (fixed → variable expense), use caching, and secure your entire infrastructure.
- Recurring AWS building blocks: CloudWatch + EC2 Auto Scaling for scaling and self-healing, ELB and message queues for loose coupling, Lambda/SQS/DynamoDB/S3/SES/Cognito for services-not-servers, and CloudFront + S3 for caching.
- The Global Infrastructure nests Region → Availability Zone → data center, with Local Zones, edge locations, and regional edge caches at the edge; Regions are isolated (no automatic cross-Region replication), and an AZ is an independent failure zone of one or more data centers.

- Choose where resources live by compliance and latency, design across multiple AZs for resiliency, and know that you pick a Region (and sometimes an AZ) but never a data center.

Review questions

1. **What is cloud architecture, and how does the house-building analogy map onto a cloud project?**

   Answer: Cloud architecture is the practice of applying cloud characteristics to a solution that uses cloud services and features to meet an organization's technical needs and business use cases. The decision maker (customer) sets the requirements, the cloud architect designs the solution (the blueprint), and the delivery team (building crew) implements it.

2. **Name the six pillars of the AWS Well-Architected Framework and one design principle of any three of them.**

   Answer: Operational Excellence (view the entire workload as code), Security (implement a strong identity foundation), Reliability (recover quickly / dynamically meet compute demand), Performance Efficiency (employ mechanical sympathy), Cost Optimization (adopt the right consumption model), and Sustainability (maximize utilization / reduce downstream impact).

3. **What does the AWS WA Tool produce, and does it change your workload for you?**

   Answer: You define a workload and answer questions across operational excellence, security, reliability, performance efficiency, and cost optimization; the tool delivers a step-by-step action plan to improve the workload. It reviews and recommends — it does not implement the changes for you.

4. **Why should trade-offs be based on empirical data, and how might you gather it?**

   Answer: Trade-offs increase the cost and complexity of an architecture, so decisions should be validated rather than assumed. Gather empirical data with load testing (to confirm a measurable performance benefit) and benchmarking (to reach the most cost-optimal workload over time).

5. **Contrast tightly coupled and loosely coupled architectures, and name the two primary decoupling solutions on AWS.**

   Answer: Tightly coupled systems chain servers directly, so one failure can be fatal and scaling forces you to reconnect every layer. Loose coupling places a managed intermediary between layers that handles failures and scaling automatically. The two primary solutions are load balancers (ELB) and message queues.

6. **Explain the difference between fixed and variable expense, and how AWS enables the shift.**

   Answer: Fixed expense is money spent to acquire, upgrade, and maintain physical assets — you pay whether or not the servers are active. AWS uses a variable-expense model: you pay only for the services you need for as long as you use them, so you provision only what you need and stop services when they are idle.

7. **Describe the AWS global infrastructure hierarchy and what makes an Availability Zone resilient.**

   Answer: Region → Availability Zone → data center, with Local Zones, edge locations, and regional edge caches at the edge. An AZ is an independent failure zone of one or more data centers, physically separated, with discrete uninterruptible power and on-site backup generation fed by different grids from independent utilities, and redundantly connected to multiple tier-1 transit providers.

8. **When you store data in one Region, is it replicated to other Regions, and can you choose the data center?**

   Answer: No on both counts. Regions are isolated and do not replicate automatically — cross-Region replication is your responsibility. You choose a Region and, for some services, an Availability Zone, but never a specific data center.