Caching Content

CACHING OVERVIEW · CLOUDFRONT CDN & EDGE CACHING · ELASTICACHE DATABASE CACHING · WELL-ARCHITECTED CACHING STRATEGIES

STUDY & REVIEW

01 MUST KNOW

if you remember five things, remember these

01 **Cache = a fast subset of data:** A **cache** is a high-speed data storage layer that *transiently* stores a subset of data so future requests are served faster, reducing the need to reach the slower underlying storage. Good candidates: **static, frequently accessed data** (HTML, CSS, JavaScript, images, video), results of **compute-intensive calculations**, and results of **slow or complex database queries** (for example, multi-table joins).

02 **Two caching types, two services: Static caching (edge caching) → Amazon CloudFront** delivers content from the nearest **edge location** as a CDN. **Database caching (data caching) → Amazon ElastiCache** serves content from an **in-memory** layer next to your database. CloudFront caches web content close to users; ElastiCache offloads read pressure from the database.

03 **CloudFront edge network + request flow: Edge locations** are numerous, close to users, with smaller caches for popular content; **Regional edge caches** are fewer, farther away, with larger caches for less-popular content and sit between your origin and the edge locations. Flow: DNS routes to the best edge → **cache hit** serves it immediately; **cache miss** fetches from the **origin** (for example, an S3 bucket), caches it, then delivers. **TTL** governs how long before content is refreshed from the origin.

04 **Control CloudFront staleness four ways:** Use them together — set a low **maximum TTL** (files expire after 24 hours by default), implement **content versioning** (new file names are fetched immediately), specify **`Cache-Control`** headers (precise per-file control), and submit **invalidation requests** (force expiry, but not immediate — use sparingly and only on individual objects).

05 **ElastiCache: engines + stale-data strategies:** ElastiCache is a fully managed, **key-value, in-memory** store with **sub-millisecond** latency, compatible with **Memcached** (simplest model, multithreaded, Auto Discovery) and **Redis** (complex data types, persistence, sorting/ranking, Multi-AZ automatic failover, pub/sub). Keep the cache fresh with **lazy loading** (populate on a cache miss — only requested data, but can go stale) or **write-through** (update the cache on every database write — always fresh, but costs more).

02 KEY TERMS

TERM	DEFINITION
Cache	High-speed data storage layer that transiently stores a subset of data so future requests are served faster than reaching primary storage
CDN (content delivery network)	Globally distributed system of caching servers that delivers a local copy of static content from a nearby edge or point of presence (PoP)
Amazon CloudFront	AWS CDN service that delivers content across the globe securely, with low latency and high transfer speeds, through edge locations
Edge location	CloudFront caching server close to users; numerous, smaller caches that serve popular content quickly
Regional edge cache	Larger CloudFront cache between the origin and edge locations; fewer and farther, holds less-popular content longer, reduces origin fetches
Origin	Server that holds the original, definitive version of your objects — an S3 bucket or a custom origin such as an HTTP/web server on EC2 or on-premises
TTL (time to live)	Setting that determines how long an edge location (or cache key) keeps content before it expires and is re-requested from the source

TERM	DEFINITION
Invalidation request	Command that forces CloudFront to expire cached content; not immediate (takes minutes) — use sparingly on individual objects
Amazon ElastiCache	Fully managed, key-value, in-memory data store/cache with sub-millisecond latency, compatible with Redis and Memcached engines
Node / cluster	A node is the smallest block of an ElastiCache deployment (network-attached RAM); a cluster is a logical grouping of one or more nodes
Lazy loading	Strategy that loads data into the cache only after it is requested (on a cache miss); caches only requested data but permits stale data
Write-through	Strategy that updates the cache immediately after the primary database is updated; keeps the cache fresh but can raise storage cost

03 HOW CACHING WORKS IN CLOUDFRONT

`TTL decides when the cached copy goes stale`

`01` User requests an object (for example, `cat.jpg`) → `02` DNS routes to the nearest / lowest-latency **edge location** → `03` **Cache hit** → CloudFront serves it immediately (done) → `04` **Cache miss** → CloudFront fetches from the **origin** (e.g., an S3 bucket) → `05` Origin returns the object to the edge location → `06` CloudFront delivers it and **adds it to the cache** for next time

04 CHOOSE AN ELASTICACHE ENGINE

`Memcached = simple · Redis = feature-rich`

ELASTICACHE FOR MEMCACHED

choose it when you need…

- The most basic model (low maintenance)
- To run large nodes with multiple cores or threads (multithreaded)
- To scale horizontally with **Auto Discovery**
- Data partitioned across up to 20 nodes

ELASTICACHE FOR REDIS

choose it when you need…

- Complex data types (strings, hashes, lists, sets, sorted sets, bitmaps)
- **Persistence** of your key store
- To **sort or rank** in-memory datasets (leaderboards)
- High availability via **Multi-AZ** with automatic failover
- Publish/subscribe messaging; scale up to 250 nodes

05 CACHING STRATEGIES: LAZY LOADING VS. WRITE-THROUGH

`match to your tolerance for stale data`

FEATURE	LAZY LOADING	WRITE-THROUGH
Caching pattern	Updates the cache **after** the data is requested	Updates the cache **immediately** after updating the primary database
Advantage	Cache holds only data the application actually requests	Cache stays current with the database — data is likely found in the cache
Disadvantage	Permits stale data; a cache miss costs a penalty (three trips)	Increases cost by storing in-memory data you might never use
Use when	Data is read often but written infrequently (e.g., a user profile)	Data must be updated in real time and always current

06 EXAM TRAPS

where the knowledge check gets you

- × “CloudFront can cache dynamic or user-generated content.” — An edge cache stores only **static** content (images, videos, web objects). CloudFront can still *deliver* dynamic content from a **custom origin**, but it does not cache it.
- × “Use CloudFront to cache your database.” — Wrong service. **CloudFront = static/edge** caching; **ElastiCache = database/in-memory** caching. On the sample exam, “CloudFront to cache from the database” is the wrong answer.
- × “Invalidation requests expire content instantly.” — They take **several minutes**, are not immediate, and should be used **sparingly on individual objects**; versioning is usually the better tool.
- × “Edge locations and Regional edge caches are the same thing.” — Edge locations are **numerous, close, small** (popular content); Regional edge caches are **fewer, farther, larger** (less-popular content) and sit **between the origin and the edge locations**.
- × “Memcached persists your cached data.” — Only **Redis** provides persistence; if a Memcached node fails or is replaced, its data is lost. Redis also adds complex data types, sorting/ranking, Multi-AZ failover, and pub/sub.
- × “Write-through is the cheap, always-right choice.” — Write-through **raises cost** by caching data you may not use. **Lazy loading** caches only requested data but permits stale data and pays a cache-miss penalty. The right pick depends on the impact of stale data.
- × “Enabling Multi-AZ on RDS fixes read-performance problems.” — Multi-AZ delivers **high availability**, not performance. To relieve heavy reads cost-effectively, add an **ElastiCache** caching layer.
- × “A cache stores a complete, durable copy of the data.” — Unlike a database, a cache **transiently** stores only a **subset** of data; it is an optimization layer, not the system of record.

07 SELF-QUIZ

answers are upside-down below

Q1 Define a cache in one sentence, and state its primary purpose.

Q2 Which AWS service provides static (edge) caching, and which provides database (in-memory) caching?

Q3 Between CloudFront edge locations and Regional edge caches, which are more numerous and closer to users, and which have the larger caches?

Q4 In the CloudFront request flow, what happens on a cache miss?

Q5 What CloudFront setting controls how long an edge location keeps content before re-requesting it, and what is the default expiry?

Q6 Name the four ways to control how long CloudFront caches content.

Q7 Which ElastiCache engine supports persistence, complex data types, sorting/ranking, Multi-AZ failover, and pub/sub messaging?

Q8 Which ElastiCache engine is multithreaded and scales horizontally with Auto Discovery?

Q9 Which caching strategy populates the cache only after data is requested, and which updates the cache immediately when the database is written?

Q10 Sample exam: an RDS database has serious performance issues from many read requests, and cost matters most. What is the best fix?

ANSWER KEY (rotate the page)

Q1 A high-speed data storage layer that stores a subset of data so future requests are served faster — it increases retrieval performance and reduces the need to reach the slower underlying storage. **Q2** CloudFront for static/edge caching; ElastiCache for database/in-memory caching **Q3** Edge locations are more numerous and closer to users; Regional edge caches have larger caches (and are fewer and farther away) **Q4** CloudFront forwards the request to the origin, the origin returns the object to the edge location, and CloudFront delivers it to the user while adding it to the cache for next time **Q5** Time to live (TTL); by default each file expires after 24 hours (the maximum TTL default) **Q6** Set a maximum TTL, implement content versioning, specify Cache-Control headers, and use invalidation requests **Q7** Redis (ElastiCache for Redis) **Q8** Memcached (ElastiCache for Memcached) **Q9** Lazy loading populates on request (a cache miss); write-through updates the cache immediately on every database write **Q10** Configure Amazon ElastiCache to cache frequently used content from the database, reducing read traffic sent to RDS