Caching Information for Scalability

01 MUST KNOW

if you remember five things, remember these

01 **Caching serves a copy, not the origin:** a **cache** is a high-speed layer that stores a **transient** copy of data; the **origin** is the authoritative source. Serving from the cache **reduces the need to access the origin**, giving faster responses, lower origin load, or both. Every cached item carries a **TTL** so it doesn't get too stale.

02 **ElastiCache = managed in-memory key-value store: fully managed**, **sub-millisecond latency**, sits **between the application and its data store**. It decreases access latency, increases throughput, eases database/application load, scales out/in/up/down, and runs the open-source **Memcached** and **Redis** engines.

03 **Memcached vs. Redis: Memcached** — multithreaded, low maintenance, Auto Discovery, **no persistence, no replication**; pick it for the **simplest model**. **Redis** — complex data types, **persistence**, replication/read replicas, Multi-AZ **automatic failover**, and **pub/sub**; pick it when you need any of those.

04 **CloudFront caches at the edge:** a **CDN** of global **edge locations (POPs)** serves content close to viewers over HTTP/HTTPS. The **cache key** (default = **distribution domain name + URL path**) decides a **hit** vs. a **miss**; files expire after **24 hours** by default, tunable with a **cache policy** or `Cache-Control`/`Expires` headers.

05 **Two caching strategies: lazy loading** updates the cache **only on a cache miss** — best when data is read often but written rarely (user profiles). **Write-through** updates the cache **whenever the database is written** — best for real-time data (leaderboards, top stories). Match the strategy to the data.

02 KEY TERMS

TERM	DEFINITION
Cache	High-speed storage layer holding a transient copy of some or all origin data; serving from it avoids requests to the origin
Origin	The original, authoritative source — a database or origin server that stores the definitive versions of the objects
Time to live (TTL)	Integer number of seconds until a cache key expires; keeps cached data from becoming too stale
Amazon ElastiCache	Fully managed, in-memory, key-value data store with sub-millisecond latency that sits between the application and its data store
Node	A fixed-size chunk of secure, network-attached RAM running one cache-engine instance; the unit you scale, each with its own DNS name and port
Cluster	A logical grouping of one or more ElastiCache nodes; every node in it is the same instance type and runs the same engine
Endpoint	The unique address an application uses to connect to an ElastiCache node or cluster
Cache hit / cache miss	Hit: requested data is in the cache and unexpired. Miss: it is absent, or the key's TTL has expired
Content delivery network (CDN)	A globally distributed system of caching servers that serves local copies of content from the nearest edge or POP
Edge location (POP)	A CloudFront data center that caches content close to viewers, connected to AWS Regions over the AWS network backbone
Cache key	The unique identifier for every object in a CloudFront cache; determines whether a viewer request is a hit

TERM	DEFINITION
Cache policy	CloudFront settings (policy info, TTL, and cache-key settings) attached to a cache behavior to control the cache key and durations

03 ELASTICACHE ENGINES — MEMCACHED VS. REDIS

know which features belong to which engine

CAPABILITY	ELASTICACHE FOR MEMCACHED	ELASTICACHE FOR REDIS
Data model	Flat strings and objects (HTML, serialized JSON)	Complex types: strings, hashes, lists, sets, sorted sets, bitmaps
Multithreading	Yes — can use more CPU cores than Redis	No
Persistence	No — data is lost when a node is replaced or scaled down	Yes — can even serve as a primary data store
Replication / HA	None — data partitioned across up to 20 nodes	Read replicas, Multi-AZ, automatic failover
Extras	Low maintenance; Auto Discovery of node changes	Publish/subscribe messaging channels
Choose when	You need the **simplest** model possible	You need persistence, advanced types, or the other features

04 LAZY LOADING VS. WRITE-THROUGH

which one keeps the cache in step with the origin

LAZY LOADING

update only on a cache miss · read-often, write-rarely (profiles)

- Only requested data is cached — **less unneeded data**
- Node failures are **non-fatal**: an empty node repopulates on misses
- Downside: the **cache-miss penalty** (three trips)
- Downside: data can go **stale** — database writes don't touch the cache

WRITE-THROUGH

update the cache on every DB write · real-time data (leaderboards)

- Data in the cache is **never stale** — always current
- Downside: **write penalty** — write to the database and the cache
- Downside: **missing data** on a new/replacement node until it is written
- Downsides: **unused data** in cache and **cache churn**

05 LAZY LOADING READ PATH (CACHE HIT / MISS)

a miss costs three trips

01 Application requests data from the cache → 02 **Cache hit** → return the cached value to the requester → 03 **Cache miss** → request the data from the origin (database) → 04 Return the value to the requester → 05 Update the cache with the value retrieved from the origin

06 EDGE FUNCTIONS — LAMBDA@EDGE VS. CLOUDFRONT FUNCTIONS

customize requests/responses at the edge

DIMENSION	LAMBDA@EDGE	CLOUDFRONT FUNCTIONS
Use case	Complex functions, full application logic	Lightweight, short-running customizations
Language	Node.js or Python	JavaScript
Event sources	Viewer request/response and origin request/response	Viewer request/response only
Built in CloudFront	No — an extension of AWS Lambda	Yes — a native CloudFront feature
Scale / limits	Greater duration, memory, and function size	Sub-ms duration, 2 MB memory, 10 KB size

07 EXAM TRAPS

where the knowledge check gets you

- × “Memcached persists data like Redis.” — Only **Redis** offers persistence (it can even be a primary data store). **Memcached has no persistence**: replace, terminate, or scale down a node and its cached data is gone.
- × “Both engines replicate for high availability.” — **Memcached has no replication** — it partitions data across up to **20 nodes**. Read replicas, Multi-AZ, and automatic failover are **Redis** features.
- × “A cache miss is one round trip.” — It is **three trips**: request the cache, query the database, then update the cache — that is the cache-miss penalty.
- × “CloudFront's cache key already includes query strings, cookies, and headers.” — By default it is **only** the distribution domain name plus the URL path. Add query strings, headers, cookies, or compression through a **cache policy**.
- × “Lambda@Edge and CloudFront Functions both run JavaScript.” — **CloudFront Functions** run JavaScript; **Lambda@Edge** runs **Node.js or Python**. Only CloudFront Functions can be built and tested entirely within CloudFront.
- × “Write-through has no drawbacks.” — Its one advantage is no stale data. It still adds a **write penalty**, leaves **missing data** on new nodes, and causes **unused data** and **cache churn**.
- × “A longer TTL is always better.” — Extending the TTL reduces misses and origin load but risks serving **stale data**; a short TTL means **more cache misses**. Best practice: set a TTL on **every** cache key.
- × “CloudFront Functions can handle origin events and full app logic.” — No — they run only on **viewer request/response** and are capped at **2 MB memory / 10 KB size**; use **Lambda@Edge** for origin events, larger workloads, or third-party libraries.

08 SELF-QUIZ

answers are upside-down below

Q1 In caching terms, what is the name for the original, authoritative source that a cache copies from?

Q2 What latency class does Amazon ElastiCache advertise, and what kind of data store is it?

Q3 Which ElastiCache engine offers persistence, replication, complex data types, and pub/sub?

Q4 Which engine do you choose when you need the simplest model possible?

Q5 What two values make up a CloudFront distribution's default cache key?

Q6 When CloudFront's cache already has the latest version of a file, what status code does the origin return?

Q7 Of Lambda@Edge and CloudFront Functions, which runs Node.js or Python, and which runs JavaScript?

Q8 Which caching strategy updates the cache only when a cache miss occurs?

Q9 Which strategy writes to the cache every time data is written to the database, so the cache is never stale?

Q10 Across how many nodes does Memcached partition data, and does it support replication?

ANSWER KEY (rotate the page)

Q1 the origin **Q2** sub-millisecond latency; a fully managed in-memory key-value store **Q3** Redis **Q4** Memcached **Q5** the distribution domain name plus the URL path of the requested object **Q6** 304 Not Modified **Q7** Lambda@Edge = Node.js/Python; CloudFront Functions = JavaScript **Q8** lazy loading **Q9** write-through **Q10** up to 20 nodes; no, Memcached has no replication