How do you define cache behaviour in CloudFront to optimize performance?

Cache behavior in CloudFront defines **how requests are handled and cached**. You control it using **policies, TTLs, and cache keys**:

🔷 **Cache Policies** – Decide what to cache (e.g., cache images but forward API calls).

🔷 **TTL (Time-to-Live)** – Defines how long objects stay cached (short TTL for HTML, long TTL for images).

🔷 **Cache Keys** – Parameters (headers, cookies, query strings) that CloudFront uses to uniquely identify cached objects.

Example:

- A React app: cache .js and .css for 24h, but API responses for only 60s (because API data changes fast).
- Use cache keys carefully: If you cache API results with query strings, each query becomes a separate cache entry.

DevOps Pipeline Use-case:

When deploying apps, pipeline scripts can update cache behaviors automatically (via Terraform/CloudFormation). For example, after pushing a new frontend release, pipeline triggers **cache invalidation + updates TTLs** so users get fresh code but images remain cached.

Which of the following best describes a Cache Policy in CloudFront?

A) Defines how long CloudFront caches objects

B) Controls which headers, cookies, and query strings are used as cache keys

C) Decides what to cache and what to forward to the origin

D) Determines where CloudFront serves traffic from

✅ **Answer: C**

Explanation: A Cache Policy tells CloudFront *what to cache and what to forward* (e.g.,

cache images but forward API calls). TTL is separate, and location is determined by Edge Locations.

In CloudFront, TTL (Time-to-Live) primarily controls:
A) Which cookies are used in the cache key
B) How long objects stay cached before revalidation
C) Which origin is selected for requests
D) Whether API responses are forwarded
✅ **Answer: B**
Explanation: TTL defines *how long objects are cached*. Example: HTML cached for 60s, images for 24h.

What happens if you include query strings in the cache key for API caching?
A) All API responses are cached under the same key
B) Each unique query string generates a separate cache entry
C) CloudFront ignores the query string and caches only one version
D) CloudFront invalidates all cache entries automatically

✅ **Answer: B**
Explanation: Cache keys use query strings to differentiate cache entries. ?id=1 and ?id=2 become separate cached objects.