Amazon CloudFront

Before we learn about Amazon CloudFront, we need to understand few terms and foundations

Content Delivery Network (CDN): A network of servers distributed geographically to deliver content more efficiently to users. CDNs help reduce latency and improve load times by caching content closer to the user's location.

Network of Edge Servers:

- A CDN consists of numerous servers (also known as edge servers) located in multiple data centres around the globe.
- These servers cache copies of your content, such as web pages, images, videos, and other types of files.

Content Caching:

- When a user requests content from a website, the CDN serves this content from the nearest edge server, reducing the distance the data has to travel.
- This caching process helps in delivering the content more quickly to users, reducing latency and load times.

Edge Location: These are data centers located around the world where CloudFront caches copies of your content. When a user requests content, it is served from the nearest edge location, reducing latency.

Origin: The origin is the source of the content that CloudFront will distribute. This can be an Amazon S3 bucket, an HTTP server (like an EC2 instance), or an Elastic Load Balancer.

Distribution: A CloudFront distribution is a configuration that tells CloudFront where your origin servers are, what content to cache, and how to manage content delivery.

Cache Behaviour: Cache behaviour settings control how CloudFront caches your content and can specify different rules for different types of content.

TTL (Time to Live): TTL is the duration that content is cached at edge locations before it is refreshed from the origin. It helps in controlling how long the content is served from the cache.

Invalidation: Invalidation is the process of removing objects from CloudFront edge caches before their TTL expires. This is used when you need to update content that has been cached.

Amazon CloudFront

AWS Global Accelerator: A networking service that improves the availability and performance of your applications with global users. It routes traffic to the optimal endpoint based on health, geography, and routing policies.

Origin Access Identity (OAI): A special CloudFront user identity used to access private content stored in Amazon S3.

Geo Restriction: A feature that allows you to restrict access to your content based on the geographic location of your viewers.

Now ...!! We will learn about the **Amazon CloudFront**

Amazon CloudFront

- **Content Delivery Network (CDN):** CloudFront is a CDN that distributes your content across a network of edge locations around the world. These edge locations cache and deliver content based on the geographic location of the end user.
- **Global Network:** CloudFront has a global network of edge locations, which are distributed across multiple geographic regions. This helps ensure that content is delivered to users with minimal latency and high performance.
- **Integration with AWS Services:** CloudFront integrates seamlessly with other AWS services such as S3 (for static file storage), EC2 (for compute resources), Lambda (for serverless compute), and API Gateway (for managing APIs).

Intro Parameters

- Global CDN that caches content at edge locations, reducing load at the origin.
- **TTL (0 sec - 1 year)** can be set by the origin using headers
- Supports **Server Name Indication (SNI)** to allow SSL traffic to multiple domains
- **CloudFront Geo Restriction** can be used to allow or block countries from accessing a distribution.
- **Edge Locations are present outside the VPC** so the origin's SG must be configured to allow inbound requests from the list of public IPs of all the edge locations.
- Supports HTTP/RTMP protocol (**does not support UDP protocol**)
- In-flight encryption using **HTTPS** (**from client all the way to the origin**)
- To block a specific IP at the CloudFront level, deploy a WAF (Web Application Firewall) on CloudFront

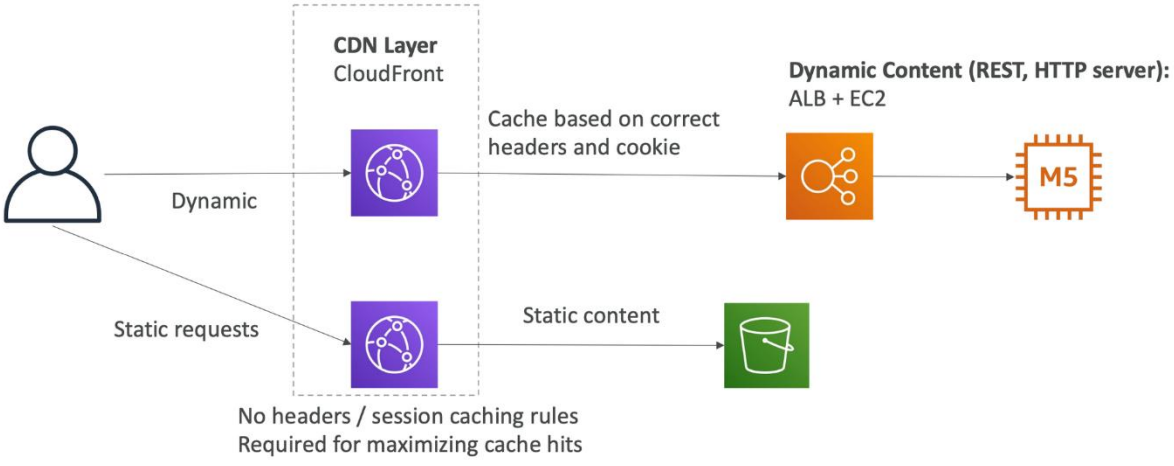
Amazon CloudFront

CloudFront supports 2 content distributions

Static Distribution & Dynamic Distribution - On high level, these are the things we need to keep in our mind

Aspect	Static Distributions	Dynamic Distributions
Purpose	Deliver content that does not change frequently.	Deliver content that is generated or customized in real-time.
Content	Static files (e.g., HTML, CSS, JavaScript, images, videos).	Dynamic data (e.g., API responses, personalized user data).
Caching	- Long-term caching at edge locations. - Efficient for reducing latency and load on origin servers.	- Cache on demand. - Forward requests to origin server when content is not cached.
Performance	- Low latency due to caching. - Reduced load on origin server.	- Can still benefit from reduced latency if cached. - Origin failover for high availability.
Cache Control	- Configurable cache duration and invalidation.	- Configurable cache behavior with forwarding of query strings, headers, and cookies.
Use Cases	- Website acceleration. - Media distribution.	- API acceleration. - Personalized content.
Example	- Hosting static assets for a website. - Serving downloadable media files.	- Delivering real-time product availability on an e-commerce site. - Providing personalized recommendations.

Example diagram



Amazon CloudFront

Pricing

- Price Class All: all regions (best performance)
- Price Class 200: most regions (excludes the most expensive regions)
- Price Class 100: only the least expensive regions

The Basic components of CloudFront is Origin, Caching, Functions, Access Control ...!!

So – we need to understand these components in-depth.

Let's start with **Origin**

In AWS CloudFront, an origin is the source location where your content is stored and from where CloudFront fetches the content to deliver to viewers. When a viewer requests content that you're serving with CloudFront, the request is routed to the nearest edge location, which then forwards the request to the origin if the content is not already cached.

Amazon S3 Bucket:

- **Static Content**: This is typically used for static content such as HTML, CSS, JavaScript, images, and videos.
- **Dynamic Content**: Amazon S3 can also be used for dynamic content, but it is less common due to latency considerations.

HTTP/S Server:

- **Web Servers**: You can use any HTTP/S server as an origin, such as an EC2 instance, an on-premises server, or another cloud provider's server.
- **Dynamic and Static Content**: Suitable for both static and dynamic content. This allows you to serve web pages, APIs, and other web applications.

Amazon EC2:

- **Custom Applications**: When you have applications running on EC2 instances, these instances can be used as origins to deliver both static and dynamic content.
- **Load Balancers**: You can also use Elastic Load Balancers (ELB) to distribute incoming application traffic across multiple EC2 instances, acting as an origin.

AWS Lambda (Lambda@Edge):

- **Custom Logic**: Lambda@Edge allows you to run custom logic (JavaScript/Node.js functions) closer to the user. This can be used to manipulate requests and responses, perform authentication and authorization, or dynamically generate content.
- **Serverless Applications**: Useful for serverless applications that need to deliver dynamic content based on user requests.

Configuring Origins in CloudFront

When setting up a CloudFront distribution, you'll need to specify one or more origins. For each origin, you can configure settings such as:

- **Origin Path**: Specifies a path that CloudFront appends to the origin domain name when forwarding requests.
- **Origin Access Control**: For Amazon S3 origins, you can set up origin access identities (OAIs) to restrict access to the S3 bucket.
- **Custom Headers**: You can add custom headers that CloudFront includes in requests to your origin.
- **Origin Shield**: Provides an additional layer of caching to help protect your origin from high request loads.

Kinds of Origin

Primary Origin

The primary origin is the main source of content that CloudFront uses to serve requests. When a request is made, CloudFront first attempts to retrieve the content from the primary origin.

Secondary Origin

The secondary origin acts as a backup to the primary origin. If CloudFront is unable to retrieve content from the primary origin (for example, due to network issues, server downtime, or other failures), it can be configured to automatically attempt to retrieve the content from the secondary origin.

- Consists of a **primary** and a **secondary** origin (can be in **different regions**)
- Automatic failover to secondary
- Provides **region-level** high availability
- Use when getting 504 (gateway timeout) error
- CloudFront routes all incoming requests to the primary origin, even when a previous request failed over to the secondary origin. It only sends requests to the secondary origin after a request to the primary origin fails.
- CloudFront fails over to the secondary origin only when the HTTP method of the viewer request is GET, HEAD, or OPTIONS.

CloudFront Functions

AWS CloudFront Functions is a feature that allows you to run lightweight JavaScript functions at the edge locations of CloudFront. These functions enable you to manipulate HTTP requests and responses, providing greater control over the content delivery process. CloudFront Functions are designed to be low-latency and highly scalable, running in less than a millisecond.

- Lightweight functions written in **JS** deployed at edge locations
- **Sub-ms startup times**
- High throughput: **millions or requests/second**
- For high scale, **latency sensitive CDN customizations**
- Can modify **Viewer Request** and **Viewer Response** only
- Native feature of CloudFront (code managed within CloudFront)
- Use cases (require < 1ms compute time):
  - URL rewrites or redirects
  - **Cache key normalization**: transform request attributes (headers, cookies, query strings, URL) to create an optimal cache key
  - **Header manipulation**: insert/modify/delete HTTP headers in the request or response
  - **Request authentication & authorization**: create and validate user-generated tokens (e.g. JWT) to allow/deny requests

Amazon CloudFront

Differences Between CloudFront Functions and Lambda@Edge

- **Execution Time:**
  - **CloudFront Functions:** Designed for very low-latency operations, running in under a millisecond.
  - **Lambda@Edge:** Suitable for more complex operations, with a higher latency allowance.
- **Programming Model:**
  - **CloudFront Functions:** Use a simplified JavaScript runtime focused on request and response manipulation.
  - **Lambda@Edge:** Supports the full AWS Lambda programming model, allowing more complex logic and access to a broader range of AWS services.

	CloudFront Functions	Lambda@Edge
Runtime Support	JavaScript	Node.js, Python
# of Requests	**Millions** of requests per second	**Thousands** of requests per second
CloudFront Triggers	- Viewer Request/Response	- Viewer Request/Response - Origin Request/Response
Max. Execution Time	< 1 ms	5 – 10 seconds
Max. Memory	2 MB	128 MB up to 10 GB
Total Package Size	10 KB	1 MB – 50 MB
Network Access, File System Access	No	Yes
Access to the Request Body	No	Yes
Pricing	Free tier available, 1/6th price of @Edge	No free tier, charged per request & duration

CloudFront Caching

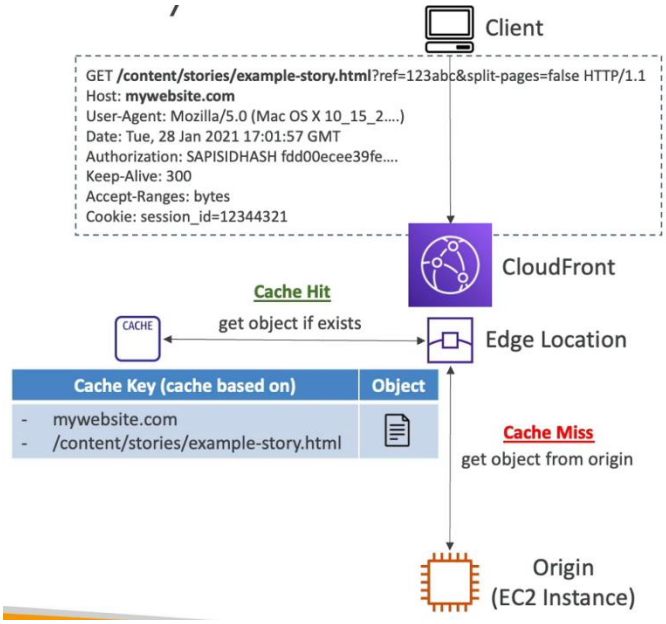
Caching in AWS CloudFront is essential for improving the performance, reliability, and efficiency of content delivery. By caching content at edge locations closer to users, CloudFront reduces latency, decreases the load on your origin servers, and provides a better user experience.

How Caching Works in CloudFront

1. **Edge Caches**:
    - **Caching at Edge Locations**: CloudFront caches your content at edge locations worldwide. When a user requests content, CloudFront checks its cache at the nearest edge location before forwarding the request to the origin.
    - **TTL (Time to Live)**: Content is cached based on the Time to Live (TTL) values you configure. TTL specifies how long the content should be cached before checking for updates from the origin.
2. **Cache Behaviors**:
    - **Customizing Cache Settings**: CloudFront allows you to define cache behaviors to specify how different types of content should be cached. You can create multiple cache behaviors for different paths, file types, or request patterns.
    - **Origin Request Policies**: Control which HTTP headers, cookies, and query strings are included in requests to the origin, helping to manage cache keys effectively.
3. **Invalidations**:
    - **Cache Invalidation**: When you update content at your origin, you might need to invalidate cached content to ensure users receive the latest version. CloudFront allows you to invalidate specific objects or entire paths in the cache.
    - **Versioning**: Using versioning in your object names (e.g., appending a version number to file names) can help avoid the need for invalidations by allowing new versions to be served immediately.

Cache Key

- **A unique identifier for each object in the cache**
- If the same cache key is created from subsequent requests, the cached content will be returned.
- **Default:** hostname + resource portion of the URL
- Can enhance the cache key (making it more complex) by adding other elements of the request (HTTP headers, cookies, query strings) using **Cache Policies**



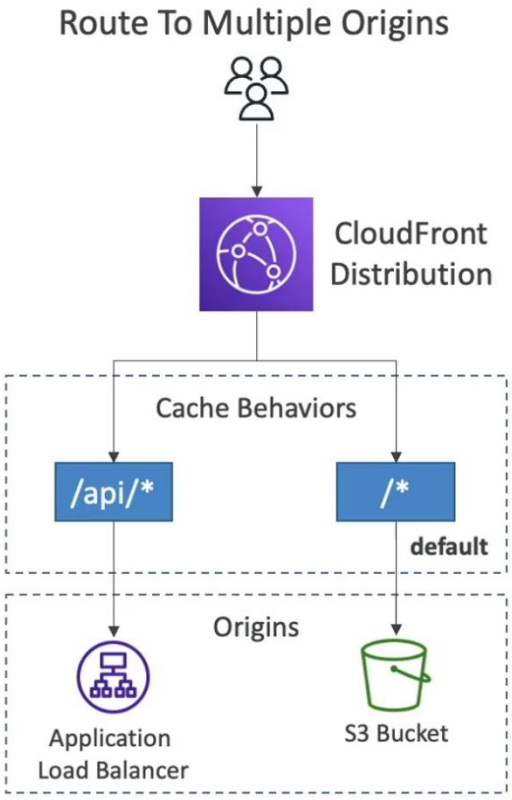
Cache Policy

- Controls how the caching mechanism works in CloudFront
- Include in cache key:
    - **HTTP Headers:** None or Whitelist
    - **Cookies:** None or Whitelist or Include All-Except or All
    - **Query Strings:** None or Whitelist or Include All-Except or All
- **The fewer items in the cache key, the better the caching performance.**
- **Controls the TTL** (can also be controlled by the origin using Cache-Control header or Expires header)
- Predefined managed cache policies or create your own
- All HTTP headers, cookies, and query strings included in the Cache Key are automatically included in origin requests

Cache Behaviors

- **Configure cache differently based on the path pattern in the request** (eg. route to different origins or origin groups based on the path pattern in the request)
- /* is the default cache behaviour (always processed at the end to find more specific pattern matches beforehand)
- **Configure different Cache Policy and Origin Request Policy for different cache behaviors**
- Validate requests differently for each cache behaviour (eg. only allow requests at /api if they have access token in the header)
- Example: Sign in page

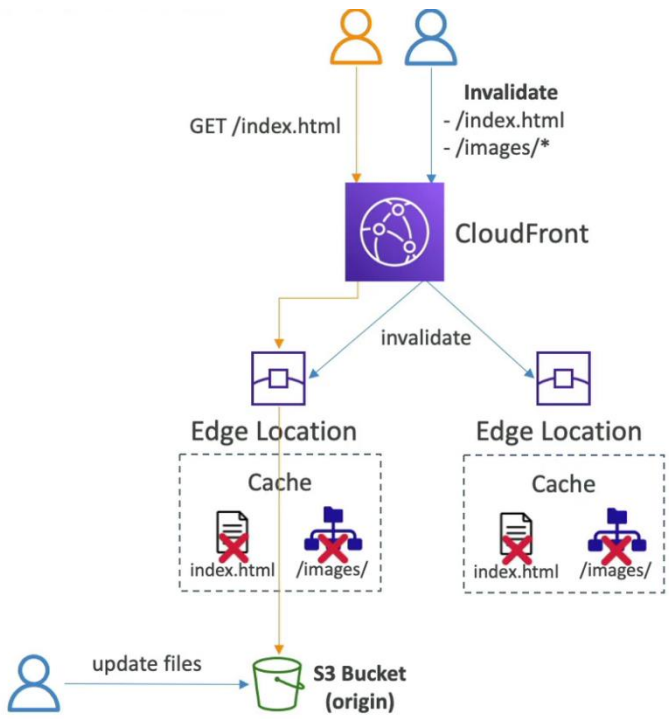
User logs in at /login and gets back signed cookies. Configure the cache policy for default cache behaviour to only accept requests if they have signed cookies.



Amazon CloudFront

Cache Invalidation

- If the content is updated at the origin, we can invalidate the cache at all the edge locations using CreateInvalidation API.
- Can invalidate the entire cache, a single file or all the files at a given path.
- **Cache invalidation is not cost-effective** (need to pay extra for invalidation requests)
- For a cost-effective solution, version your objects using the path or filename and update the application to pull the new version.



Access Control

Signed URLs

Signed URLs in AWS CloudFront are used to control access to your content, ensuring that only authorized users can access it. This is particularly important for scenarios where you need to protect sensitive or premium content, enforce time-limited access, or integrate with subscription or pay-per-view models.

How Signed URLs Work

1. **URL Creation**:

Generate Signed URL: A signed URL is generated by creating a URL that includes additional query parameters such as the expiration time, signature, and possibly other custom parameters.

Signing Process: The URL is signed using a private key associated with a CloudFront key pair. The signature is a hash of the URL and parameters, encrypted with the private key.

2. **URL Distribution**:

Distribute Signed URL: The signed URL is distributed to authorized users, either through a secure application, email, or other means.

3. **URL Validation**:

CloudFront Validation: When a user requests content using the signed URL, CloudFront validates the signature and checks the expiration time. If the URL is valid and not expired, CloudFront serves the content.

Access Denial: If the URL is invalid or expired, CloudFront denies access and returns an appropriate error response.

Configuring Signed URLs in CloudFront

1. **Create a Key Pair:**

In the AWS Management Console, create a CloudFront key pair. This consists of a public key (stored with CloudFront) and a private key (kept secure by you).

2. **Configure CloudFront Distribution:**

Associate the public key with your CloudFront distribution. This enables CloudFront to verify signed URLs using the public key.

3. **Generate and Use Signed URLs:**

Use the private key to generate signed URLs for content that you want to protect. Distribute these URLs to authorized users.

Amazon S3 Pre-Signed URLs

S3 pre-signed URLs provide temporary, secure access to individual objects stored in Amazon S3. They are generated to grant time-limited access to private S3 objects.

Feature	S3 Pre-Signed URL	CloudFront Signed URL
Access Method	Direct access to S3 objects	Access via CloudFront edge locations
Primary Use Case	Temporary access to S3 objects	Secure delivery of content via CDN
Expiration	Short-lived (defined at creation)	Customizable expiration
Performance	Direct S3 access, no CDN caching benefits	Benefits from CDN caching and edge locations
Generation	S3 API/SDK	CloudFront API/SDK
Application	Single S3 object	Any object within a CloudFront distribution
Security	Temporary access without changing permissions	Enhanced access control via signed URLs
Complexity	Simpler to implement	More complex, involves signing with key pair

Choosing Between S3 Pre-Signed URLs and CloudFront Signed URLs

- **Use S3 Pre-Signed URLs** if you need to provide temporary, direct access to specific S3 objects, especially for short-term access or uploads/downloads.
- **Use CloudFront Signed URLs** if you need to control access to content delivered through CloudFront, taking advantage of edge location caching, enhanced performance, and more complex access control mechanisms.

Encryptions

Field-level Encryption

- Sensitive information sent by the user is encrypted at the edge close to user which can only be decrypted by the web server (intermediate services can't see the encrypted fields)
- **Asymmetric Encryption** (public & private key)
- Max 10 encrypted field

TLS Encryption

- **Origin Protocol Policy**: used to enable SSL between the distribution and the origin
- **Viewer Protocol Policy**: used to enable SSL between the client (user) and the distribution
- TLS termination takes place at the distribution level. If the distribution - origin connection needs to be encrypted, another TLS connection is established.

Numerical Questions

https://lisireddy.medium.com/aws-cloudfront-numerical-questions-0b453bc5d2f1

Scenario based Questions

https://lisireddy.medium.com/aws-cloudfront-scenario-based-questions-0c28a460a71e

Wish you the best …! Happy Learning ..!

Yours' Love (@lisireddy across all the platforms)

Amazon
CloudFront