**AWS CloudFront:**

When delivering content globally, your web server may be situated in one region (for example, North America). If a user in India sends a request, the data must travel a long distance, resulting in high latency. To mitigate this, AWS uses edge locations: smaller, geographically dispersed sites that cache data from your origin (such as a web server or an S3 bucket). Users receive content from the nearest edge location, ensuring minimal delay.

For example, if users in Australia access your application, they connect to a local edge location instead of the distant origin server. This proximity reduces the interaction time with your application, making it more responsive. Essentially, Amazon CloudFront improves performance by caching content closer to users.

## **What is CloudFront?**

CloudFront is a web service that accelerates the distribution of both static and dynamic content—including HTML, CSS, JavaScript, images, videos, and music—by delivering it from the closest edge location. This global network of edge locations minimizes latency by serving cached copies of your content rather than always querying a centralized server.

### **Origin**

The origin is the source of the content that CloudFront caches. This can be an S3 bucket storing images and files, or a custom origin like a load balancer or an HTTP server running on an EC2 instance. Once CloudFront fetches the content from the origin, it caches the data at nearby edge locations.

### **Distribution**

A distribution in CloudFront is a configuration block where you define the origin settings. CloudFront generates a unique domain name (e.g., xyz.cloudfront.net) that users leverage to access cached content. Upon receiving a request, CloudFront checks for a cached version at the closest edge location. If available, it serves the file immediately; otherwise, it retrieves the content from the origin, caches it, and then delivers it to the user.

### **Time to Live (TTL)**

The Time to Live (TTL) defines how long content remains cached at an edge location. By default, the TTL is set to 24 hours, meaning content is served from the cache for that duration before it is considered stale. If a user requests the content after the TTL expires, CloudFront must fetch a fresh copy from the origin and update its cache. You have the flexibility to modify the TTL based on your content freshness requirements.

### **Cache Invalidation**

Sometimes you need to update content before the TTL expires. CloudFront enables manual cache invalidation to remove outdated content from all edge locations. For example, if you update a file (from version one to version two), users might still retrieve the outdated version until the TTL lapses. By initiating a cache invalidation, CloudFront removes the old version so that the new version is fetched on subsequent requests.

Invalidations can be applied at the distribution level using a wildcard (e.g., "/\*"), specific directories, or individual file paths.

## **Integration with Other AWS Services**

CloudFront integrates seamlessly with several AWS services to enhance its functionality:

* **SSL/TLS:** HTTPS is enabled by default. AWS provides a default SSL certificate (e.g., \*.cloudfront.net), and you can use AWS Certificate Manager to customize certificates for your domains.
* **CloudWatch:** CloudFront automatically pushes operational metrics to CloudWatch, allowing you to monitor performance. You can also opt for additional metrics at an extra cost.

## **Use Cases for CloudFront**

CloudFront is versatile and supports a wide range of real-world use cases:

| Use Case | Description | Example |
| --- | --- | --- |
| Static Websites | Ideal for hosting websites without dynamic server-side logic. | Delivering HTML, CSS, and JavaScript files. |
| Video on Demand | Efficiently caches and delivers video content for on-demand streaming. | Streaming pre-recorded video content to users. |
| Media File Delivery | Ensures fast distribution of images, documents, and other media libraries. | Distributing downloadable files such as PDFs or images. |

## **Summary**

Amazon CloudFront leverages a global network of edge locations to deliver content with reduced latency by caching files closer to users. The core components include:

* Origin: The source of your content.
* Distribution: The configuration that defines how content is delivered.
* TTL: The duration for which cached content remains valid (default: 24 hours).
* Cache Invalidation: The process to manually refresh outdated content.

**CloudFront Demo**

## **Setting Up the Origin (S3 Bucket)**

Before configuring CloudFront, we need an origin to store our files. Although any web server, load balancer, or API endpoint can be used as an origin, this demo uses an Amazon S3 bucket for simplicity.

1. Create a New S3 Bucket  
   Open the [Amazon S3 console](https://aws.amazon.com/s3/) and create a new bucket and leave the default region settings. Ensure that the bucket is configured to allow internet access.
2. Upload an Image File  
   Open the created bucket and upload the file car.jpg (a blue car image) by dragging and dropping it into the bucket.

Verify Object Access  
After uploading, click on the file to view the object URL. If you encounter an "Access Denied" error as shown below, it indicates that the bucket policy hasn’t been configured for public access.

| <**Error**>  <**Code**>AccessDenied</**Code**>  <**Message**>Access Denied</**Message**>  <**RequestId**>LEGJQIT1HU0Z8N1X2P1F6X1H15/Requests15</**RequestId**>  <**HostId**>86u4cayk5wB7dEFlCkNFnKf2dC5D1E7UY2c0/38Vnty6toGvYwXlP8iM8WnDZe</**HostId**> </**Error**> |
| --- |

The error occurs because the S3 bucket policy restricts public access by default.

**Configure the Bucket Policy**  
 To allow public access, navigate to the bucket's permissions and add the following JSON policy. (Remember to update the bucket name if needed.)

| {  "version": "2012-10-17",  "statement": [  {  "sid": "PublicReadGetObject",  "Effect": "Allow",  "Principal": "\*",  "Action": [  "s3:GetObject"  ],  "Resource": [  "arn:aws:s3:::bucket\_name/\*"  ]  }  ] } |
| --- |

Confirm Public Accessibility  
After applying the policy, clicking the object URL should display the image successfully.

## **Configuring CloudFront**

With the origin set up, the next step is to configure CloudFront to cache the image at edge locations for enhanced performance.

1. **Create a CloudFront Distribution**  
   In the CloudFront console, create a new distribution. Under "Origin Domain," select your S3 bucket (bucket\_name). If you wanted to cache a specific folder (for example, /images), you could enter that in "Origin Path." For this demo, leave the origin path blank to cache all objects.
2. Adjust Distribution Settings  
   Update the following settings as needed:
   * **Origin Access:**  
     Set to public if you want users to access the S3 URL directly, or configure origin access control for enhanced security by limiting access exclusively through CloudFront.
   * **Compress Objects Automatically:**  
     Set this option to Yes for performance improvements.
   * **Allowed Protocols:**  
     Enable both HTTP and HTTPS. For a production environment, it's recommended to enforce HTTPS only.
   * **Allowed HTTP Methods:**  
     For static content, GET is sufficient. Additional methods (PUT, POST, PATCH) can be enabled if required.
   * **Edge Locations:**  
     CloudFront uses all edge locations by default; you can modify this to restrict caching to specific regions if necessary.
3. Other settings like AWS Certificate Manager for certificates and IPv6 support can remain at their default values for this demonstration.
4. **Deploy the Distribution**  
   Once configured, create the distribution. Deployment may take a few minutes. When it’s complete, the distribution shows as enabled and displays a domain name you can use to access the cached files.
5. **Access the Cached Image**  
   To test the configuration, enter your distribution's domain name in the browser followed by /car.jpg. Note that accessing the domain root will not work since the S3 bucket is configured for static objects without an index.

At this stage, your S3-hosted image should load via CloudFront’s edge locations, delivering faster content to users.

## **Demonstrating Cache Behavior with Invalidation**

After verifying that CloudFront is serving the cached blue car image, we will update the object in the S3 bucket to demonstrate CloudFront's caching behavior and invalidation process.

1. Update the Image in S3
   * Delete the existing car.jpg from the S3 bucket.
   * Upload a new image (a red car) with the same file name (car.jpg). When accessing the direct S3 URL, you should now see the red car image.
2. **Observe Cache Persistence**  
   Refresh the CloudFront distribution URL for /car.jpg. You may still see the blue car image because it is cached with a default TTL (Time to Live) of 86,400 seconds (24 hours).

**Caching Behavior**

The cached content persists until the TTL expires. If immediate updates are required, you must invalidate the cache.

Invalidate the Cache  
To force CloudFront to fetch the updated image before the TTL elapses, create an invalidation request:

* + In the CloudFront console, select your distribution and go to the "Invalidations" tab.
  + Create a new invalidation. To invalidate a specific file, enter /car.jpg. Alternatively, to invalidate all objects, use /\*. You can also invalidate a folder using a pattern like /images/\*.

1. Verify the Invalidation  
   Once the invalidation process is complete, refresh the CloudFront URL for /car.jpg. The red car image should now appear as CloudFront fetches the updated object from the S3 bucket.
2. Review TTL and Caching Policy  
   For further insights, check the TTL and caching policy by navigating to the "Behaviors" tab in your CloudFront distribution settings and clicking "Edit" on the appropriate behavior.

## **Conclusion**

## we demonstrated how to set up an Amazon S3 bucket as the origin for static content and configure an AWS CloudFront distribution to cache that content at edge locations. We also covered how to perform cache invalidation to ensure that updates propagate before the default TTL expires. This flexible setup is also ideal for hosting static websites, ensuring quick and reliable content delivery.

Happy caching, and enjoy building faster web experiences!